VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Komponenta výukového serveru TI - Predikátová logika

Component of Educational Server for Theoretical CS - Predicate Logic

2019 Michaela Lubyová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Michaela Lubyová**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Komponenta výukového serveru TI - Predikátová logika
Component of Educational Server for Theoretical CS - Predicate Logic

Jazyk vypracování: čeština

Zásady pro vypracování:

V rámci diplomových a bakalařských prací vzniká výukový server pro předměty teoretické informatiky. Jedná se o sadu dynamických webových stránek umožňujících studentům pochopení různých typů úloh a problémů tím, že si mohou zadat na stránce libovolné zadání a zobrazí se jim řešení včetně postupu. Cílem této diplomové práce je vytvořit komponentu pro výuku predikátové logiky 1. řádu.

1. Vytvořte dynamické webové stránky umožňující uživateli následující:
a) zadat řetězec a nechat si syntakticky zkontrolovat, zda jde o formuli predikátové logiky,
b) zobrazit syntaktický strom zadané formule,
c) nechat si automaticky převést formuli do běžně používaných speciálních tvarů - prenexní tvar kvantifikátorů, eliminace ekvivalence a implikace apod.,
d) pro některé interpretace si nechat automaticky vyhodnotit, zda je při dané interpretaci formule pravdivá,
e) interpretace budou podporovat libovolná malá konečná univerza spolu s libovolnými relacemi a funkcemi a případně několik nekonečných spolu s předdefinovanými funkcemi a relacemi.
2. Vytvořte alespoň 5 ukázkových formulí a ke každé minimálně 2 interpretace tak, aby v jedné byla a ve druhé nebyla pravdivá.
3. Použité technologie budou voleny tak, aby byly stránky na straně klienta co nejméně platformně závislé.

Seznam doporučené odborné literatury:

[1] M.Duží: Logika pro informatiky (a příbuzné obory), Ostrava, 2012
[2] M. Huth and M. Ryan: Logic in Computer Science, Cambridge University Press, 2004

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Kot, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prehlasujem, že som túto diplomovú prácu vypracovala samostatne. Uviedla som všetky literárne pramene a publikácie, z ktorých som čerpala.

V Ostrave 29. apríla 2019 .....

Chcela by som na tomto mieste poďakovať svojmu vedúcemu Ing. Martinovi Kotovi, Ph.D. za jeho cenné postrehy a čas venovaný konzultáciám. Ďalej by som chcela poďakovať svojej rodine a priateľovi za podporu a pochopenie.

Abstrakt

Cieľom tejto diplomovej práce je vytvorenie webovej aplikácie zameranej na prácu s formulami predikátovej logiky. Aplikácia má byť schopná syntakticky analyzovať reťazce za účelom zistenia, či sa jedná o formule predikátovej logiky, vytvoriť z formúl syntaktické stromy, previesť formule do prenexovej normálnej formy a vyhodnotiť pravdivosti formúl na základe interpretácií. V diplomovej práci je popísaná teória súvisiaca s týmito operáciami, technológie použité pre vývoj aplikácie a taktiež aj jej samotná implementácia.

Kľúčové slová: webová aplikácia, výukový server, predikátová logika, syntaktická analýza, syntaktický strom, prenexová normálna forma, pravdivosť formúl, teoretická informatika

Abstract

The goal of this master's thesis is to create a web application focused on working with first-order logic formulas. The application should be able to syntactically analyze strings in order to find out if they are first-order logic formulas, create parse trees from formulas, transform formulas into the prenex normal form and evaluate formulas' truth values based on interpretations. In the master's thesis, there is described the theory related to these operations, technologies used for developing the application and also its implementation.

Key Words: web application, educational server, first-order logic, syntax analysis, syntax tree, prenex normal form, truth values, theoretical computer science

Obsah

Zoznam použitých skratiek a symbolov

KNF	–	Konjunktívna normálna forma
DNF	–	Disjunktívna normálna forma
PNF	–	Prenexová normálna forma
UML	–	Unified Modeling Language
HTML	–	Hyper Text Markup Language
ASP	–	Active Server Pages
HTML	–	Hyper Text Markup Language
XML	–	eXtensible Markup Language
VŠB-TUO	–	Vysoká škola báňská-Technická univerzita Ostrava

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov zdrojového kódu

1 Úvod

Predikátová logika je odvetvím logiky, ktorého cieľom je poskytovať formálnu reprezentáciu tvrdení a umožňovať jednoduchšiu prácu s nimi. Z tvrdení v prirodzenom jazyku, s využitím formálneho jazyka, sú vytvárané správne vytvorené formule. S formulami predikátovej logiky je možné pracovať rôznymi spôsobmi a väčšinou je k tomu potrebné poznať viacero algoritmov a zákonov. Preto bola v rámci tejto diplomovej práce vyvinutá webová aplikácia, ktorá je schopná niektoré operácie s formulami vykonávať automaticky. Webová aplikácia ponúka používateľom pomoc pri riešení a pochopení niekoľkých typov úloh, ktoré sa týkajú práce s formulami predikátovej logiky. Táto diplomová práca poskytuje popis vytvorenej webovej aplikácie a taktiež zhrnutie teórie potrebnej k pochopeniu operácií, ktoré vykonáva.

Kapitola 2 tejto diplomovej práce hovorí o logike vo všeobecnosti. Zavádza pojem správneho usudzovania, správneho úsudku a definuje jeho hlavnú myšlienku. Ďalej popisuje jedno konkrétne odvetvie logiky, formálnu logiku, formálny jazyk a prvky, ktorými je tvorený.

Kapitola 3 je zameraná na výrokovú logiku. Na začiatku definuje základný prvok výrokovej logiky, výrok, a jeho dva hlavné typy. V ďalšej časti sa zaoberá formálnym jazykom výrokovej logiky, pomocou ktorého je možné urobiť z výroku správne vytvorenú formulu. Popisuje abecedu a gramatiku formálneho jazyka, a taktiež uvádza príklad prevodu výroku z prirodzeného jazyka do jazyka formálneho. V poslednej podkapitole je definovaná sémantika formúl výrokovej logiky, teda ich význam. Taktiež je v nej predstavený spôsob vyhodnocovania pravdivosti formúl a vysvetlenie významu jednotlivých logických spojok.

Kapitola 4 sa zaoberá ďalším odvetvím logiky, a to predikátovou logikou. V úvodnej časti hovorí o rozdieloch medzi výrokovou a predikátovou logikou, a definuje typy tvrdení, ktoré výroková logika nie je schopná správne formalizovať. Potom popisuje abecedu a gramatiku svojho formálneho jazyku, a taktiež uvádza príklad prevodu výroku z prirodzeného jazyka do formálneho jazyka. Na záver vysvetľuje pojem semántiky v predikátovej logike a akým spôsobom sa vyhodnocuje pravdivostná hodnota formúl na základe interpretácií.

V kapitole 5 sú popísané ekvivalentné úpravy, pomocou ktorých je možné prevádzať formule výrokovej aj predikátovej logiky do ekvivalentných tvarov. Je v nej definovaných niekoľko druhov zákonov, ktorých základom je ekvivalencia medzi formulami.

Kapitola 6 je zameraná na špeciálne tvary formúl. Konkrétne predstavuje komplexnejšie ekvivalentné úpravy zložené z niekoľkých jednoduchších úprav opísaných v kapitole 5. základnými špeciálnymi tvarmi, do ktorých je možné previesť formule výrokovej logiky, sú konjunktívna a disjunktívna normálna forma. Pre formule predikátovej logiky je ňou prenexová normálna forma.

V kapitole 7 je predstavený zámer webovej aplikácie, technológie, ktoré boli použité pri jej vývoji, spôsob jej nasadenia na server a diagramy, ktoré reprezentujú informácie o nej.

Kapitola 8 sa zaoberá samotnou implementáciou webovej aplikácie. Popisuje, akým spôsobom boli implementované jednotlivé časti aplikácie, teda syntaktická kontrola, vytvorenie

syntaktického stromu, prevod do prenexovej normálnej formy a vyhodnocovanie pravdivosti formúl na základe interpretácií. Taktiež ukazuje správanie používateľského rozhrania pre konkrétne zadané vstupy.

Kapitola 9, záver, sa zaoberá cieľom diplomovej práce, teda vytvorením webovej aplikácie zameranej na prácu s formulami predikátovej logiky. V skratke uvádza vývojové prostredie a technológie, ktoré boli použité pri jej vývoji. Popisuje, čím je tvorené používateľské rozhranie aplikácie a ako je možné ho používať. Na záver hovorí o možnosti využitia aplikácie ako komponenty výukového serveru pre teoretickú informatiku a jej rozšírení v budúcnosti.

2 Logika

Logika je veda, ktorá sa zaoberá štúdiom ľudského myslenia, jeho analýzou, ale najmä problematikou správneho *usudzovania* [1]. Usudzovanie je chápané ako vyvodzovanie dôsledkov z nejakých tvrdení a jeho základným prvkom je úsudok.

Úsudkom sa označuje taký myšlienkový postup, ktorým je možné dospieť od nejakých tvrdení - predpokladov, nazývaných aj premisy, k inému tvrdeniu - dôsledku, záveru. Ak z pravdivých predpokladov vždy získame pravdivý záver, tak je úsudok nazývaný logicky správnym alebo platným, a teda môžeme tvrdiť, že vyvodený záver z daných predpokladov vyplýva. Inými slovami, ak je úsudok platný tak nie je možné, aby predpoklady boli pravdivé a zároveň dôsledok nepravdivý.

Logika sa, od svojho vzniku, vyvíjala a vznikali z nej rôzne odvetvia. V rámci informatiky sa pracuje hlavne s odvetvím nazývaným formálna, matematická alebo symbolická logika. Táto logika sa zaoberá logickou štruktúrou tvrdení a vzájomnými vzťahmi medzi nimi. Okrem toho, definuje *formálny jazyk* s vlastnou syntaxou a sémantikou, do ktorého je možné previesť tvrdenia z prirodzeného jazyka [2].

Formálny jazyk, taktiež označovaný aj symbolický, je jazyk, ktorého hlavnou úlohou v oblasti logiky je sprehľadnenie a zjednodušenie práce s tvrdeniami. K tomuto cieľu sa dostáva pomocou rozumnej a jednotnej reprezentácie tvrdení. Túto reprezentáciu definujú abeceda a gramatika formálneho jazyka.

Abeceda je neprázdna množina symbolov, ktorými je možné tvrdenia reprezentovať. Gramatika je neprázdna množina pravidiel, podľa ktorých je možné určiť či je slovo správne vytvorené alebo nie, a teda zistiť, či slovo do formálneho jazyka patrí. Najznámejšími odvetviami formálnej logiky sú *výroková logika* a *predikátova logika*, ktorými sa budú zaoberať viac do hĺbky kapitoly 3 a 4.

3 Výroková logika

Výroková logika je odvetvie formálnej logiky, ktorého základným prvkom je *výrok*. Študuje logické vzťahy medzi jednotlivými výrokmi a spôsoby, akými je možné spájať jednoduchšie výroky do zložitejších [3]. Nezaoberá sa, však, obsahom ani vnútornou štruktúrou samotných najmenších výrokov. S každým elementárnym výrokom pracuje ako s významovo nezávislým na ostatných. Z toho vyplýva, že vlastný zmysel výroku nie je vo výrokovej logike pre správne usudzovanie podstatný.

Výrokom sa nazýva každé tvrdenie, o ktorom má zmysel prehlásiť či je pravdivé alebo nepravdivé, pričom môže nastať práve jedna z týchto možností. Tvrdením v prirodzenom jazyku môže byť len jeden typ vetry, a to veta oznamovaciam keďže rozkazu, žiadosti ani zvolaniu nie je možné určiť pravdivosť.

Výroky sa delia na *jednoduché* (atomické, elementárne) a *zložené*. Jednoduchý výrok je taký, ktorý je z hľadiska výrokovej logiky ďalej nedeliteľný, čiže žiadna jeho vlastná časť nie je sama o sebe výrokom. Takýto výrok neobsahuje žiadnu logickú spojku a je braný ako celok. Príkladmi atomických výrokov sú „V Banskej Bystrici sneží.“ alebo „Vysoké Tatry sú najvyšším pohorím na Slovensku.“.

Spojením jednoduchých výrokov dohromady pomocou logických spojok vzniká výrok zložený, teda výrok, ktorý obsahuje aspoň jednu vlastnú časť, ktorá je tiež výrokom. Zložené výroky je taktiež možné ďalej spájať pomocou logických spojok do viac a viac komplexnejších výrokov. Príkladom zloženého výroku je, napr. „Nie je pravda, že ak svieti slnko, tak hviezdy neexistujú.“ alebo „V Ostrave prší, práve vtedy, keď nemám so sebou dáždnik, alebo keď mám obuté premokavé topánky.“.

3.1 Formálny jazyk výrokovej logiky

Prirodzený jazyk nie je pre dokazovanie výrokov ideálny. Výroky zvyčajne pozostávajú z niekoľkých slov, nie len z jedného, čo ich robí menej prehľadnými. Zároveň, ako už bolo vyššie spomínané, ich význam nie je pre výrokovú logiku podstatný. Preto sa zavádza pojem formálneho jazyka, do ktorého je možné výroky z prirodzeného jazyka prevádzať, a tým prácu s nimi zjednodušiť a sformálniť.

Formálny jazyk výrokovej logiky označuje neprázdnu množinu slov takú, že každé jej slovo je tzv. *správne vytvorenou formulou* (ďalej len formula) [3]. Tieto formule, teda slová formálneho jazyka, sú konečné postupnosti znakov, reťazce, definované abecedou a gramatikou. Abeceda jazyka je tvorená množinou symbolov, z ktorých sa vytvárajú formule výrokovej logiky. Sú nimi tieto tri druhy symbolov:

1. výrokové symboly:
   - písmená malej abecedy, prípadne s indexmi (p,q,...)
   - písmená veľkej abecedy, prípadne s indexmi (A,B,...)
2. symboly pre logické spojky:
   - $\neg$ - negácia („nie je pravda, že“)
   - $\wedge$ - konjunkcia („a“)
   - $\vee$ - disjunkcia („alebo“)
   - $\rightarrow$ - implikácia („ak...,tak“)
   - $\leftrightarrow$ - ekvivalencia („práve vtedy, keď“)
3. pomocné symboly: zátvorky ( ), prípadne [ ]

Výrokové symboly, inak nazývané aj výrokové premenné, predstavujú jednotlivé atomické výroky. Logické spojky reprezentujú spojky z prirodzeného jazyka. Ich významy sú popísané v zátvorkách nachádzajúcich sa pri nich. Slovné spojenia v zátvorkách nie sú jediné, ktoré je možné zameniť za logické spojky. Existujú aj iné alternatívy, ale významovo musia znamenať to isté, napr. ekvivalencia môže byť vyjadrená aj ako „vtedy a len vtedy, keď“. Pomocné symboly, teda zátvorky, sa v zápisoch mnohých formúl používať nemusia, ale častokrát sa využívajú už len kvôli prehľadnosti. Prípady, kedy sa zátvorky používať musia a kedy nie sú popísané v podkapitole 3.1.1.

Aby sa slovo stalo formulou výrokovej logiky, musí byť tvorené výhradne zo znakov spomínanej abecedy a musia ho definovať pravidlá, ktoré popisuje gramatika formálneho jazyka. Sú nimi nasledujúce pravidlá:

1. každý výrokový symbol je atomická formula
2. každá atomická formula je formula
3. ak A je formula, tak je formulou aj (A)
4. ak A je formula, tak je formulou aj $\neg A$
5. ak A a B sú formule, tak sú formulami aj $A \wedge B$, $A \vee B$, $A \rightarrow B$, $A \leftrightarrow B$

Atomické formule, taktiež nazývanú aj elementárne, neobsahujú žiadny podreťazec, ktorý by bol sám formulou. To znamená, že neobsahujú žiadnu podformulu. Formule, ktoré nejakú podformulu obsahujú, sa nazývajú zložené formule. Zložené formule vznikajú pomocou pravidiel 3, 4 a 5.

3.1.1 Prevod výroku z prirodzeného jazyka do jazyka výrokovej logiky

Pre ukázanie prevodu výroku z prirodzeného jazyka do formálneho jazyka výrokovej logiky, si zoberieme výrok spomenutý v kapitole 3, konrétne „V Ostrave prší, práve vtedy, keď nemám so sebou dáždnik, alebo keď mám obuté premokavé topánky.“. V prvom kroku musíme identifikovať výrokové premenné a priradiť im význam. V tomto konkrétnom prípade ich môžeme zvoliť nasledovne:

- o: v Ostrave prší
- d: mám so sebou dáždnik
- p: mám obuté premokavé topánky

Potom zameníme slovné spojenia „nie je pravda, že“, „a“, „alebo“, „ak. . . ,tak“, „práve vtedy, keď“ za odpovedajúce logické spojky. Niektoré spojky, však nemusia byť v tvrdení dané explicitne a je potrebné ich vydedukovať z kontextu. Nasledujúcim krokom je pospájanie symbolov do reťazca podľa pravidiel gramatiky. Z reťazca sa tak stane slovo formálneho jazyka výrokovej logiky, teda formula.

Pri spájaní symbolov do reťazca je taktiež možné použiť aj pomocné symboly, konkrétne zátvorky. V niektorých prípadoch ich nie je potrebné použiť, ale môžu sa pre lepšiu prehľadnosť. V iných prípadoch je ich použitie nutné, a to hlavne vtedy, keď je potrebné uprednostniť nejaký operátor pred iným kvôli významu formuly. Obsahy zátvoriek sa vyhodnocujú skôr ako zvyšok formule, konkrétne v poradí od najvnútornejšej zátvorky po najvonkajšiu, čo umožňuje meniť prioritu operátorov. Zoradenie operátorov podľa priority [4], od najvyššej po najnižšiu, vyzerá nasledovne:

1. $\neg$
2. $\wedge$
3. $\vee$
4. $\rightarrow, \leftrightarrow$

Ak teda chceme, aby sa operátor vyhodnotil skôr ako iný z vyššou prioritou, potrebujeme použiť zátvorky.

Z príkladu, ktorý sme si na začiatku tejto podkapitoly určili, získame po prevode do formálneho jazyka formulu: $o \leftrightarrow (\neg d \vee p)$. Ak by sme nepoznali význam výrokových symbolov, tak by táto formula mohla predstavovať veľa iných možných výrokov, napr. „Električka číslo 8 príde neskoro na zastávku Stodolní, práve vtedy, keď vonku nie je pekné počasie, alebo keď sa ponáhľam.“ alebo „Potrebujem niečo vytlačiť na tlačiarni práve vtedy, keď nie je funkčná, alebo keď sa minie toner.“.

3.2 Sémantika jazyka výrokovej logiky

Sémantika, alebo význam, formúl výrokovej logiky je daná ich interpretáciou. Interpretácia je funkcia, ktorá priraďuje význam symbolom formálneho jazyka. Vo výrokovej logike priraďuje význam výrokovým premenným, ktoré reprezentujú elementárne výroky.

Formula v danej interpretácii môže byť buď pravdivá, alebo nepravdivá, nič medzi tým. Atomická formula je interpretovaná tak, že za výrokový symbol je dosadená jedna z hodnôt pravda alebo nepravda, čím získame dve rôzne interpretácie jednej formule. Keďže elementárne formule sú najmenšie možné formule, tak z toho vyplýva, že každá formula môže byť interpretovaná minimálne dvomi rôznymi spôsobmi. Týchto spôsobov však môže byť aj oveľa viac, v niektorých prípadoch až nekonečne veľa.

Vo všeobecnosti, vo výrokovej logike je možné interpretovať ľubovoľnú formulu aplikovaním nasledujúceho postupu. V prvom kroku sa každej výrokovej premennej v danej celkovej formule, priradí jedna z dvoch pravdivostných hodnôt, pravda alebo nepravda. V druhom kroku sa na interpretáciou priradené hodnoty výrokových symbolov aplikujú *logické operácie*, ak sa vo formule nejaké nachádzajú.

Logická operácia je taký druh operácie, ktorej vstupnými argumentmi sú pravdivostné hodnoty, teda logická 1 - pravda alebo logická 0 - nepravda, a výstupom je tiež pravdivostná hodnota 1 alebo 0, ktorá závisí od zadaných vstupov a konkrétnej operácie.

Tieto operácie sú vo formálnej logike reprezentované pomocou logických spojok $\neg$, $\wedge$, $\vee$, $\rightarrow$ a $\leftrightarrow$, z ktorých každá spojka predstavuje jednu operáciu. Operácia negácie je jediná unárna, teda ako vstup berie jeden argument, ostatné sú binárne, čiže potrebujú 2 vstupné argumenty. Všetky možné kombinácie vstupov logických operácií, a z nich vyplývajúce výstupy, sa zapisujú do pravdivostných tabuliek [4], viď Tabuľka 1 a Tabuľka 2.

A	$\neg$A
1	0
0	1

Tabuľka 1: Pravdivostná tabuľka pre negáciu

A	B	A$\wedge$B	A$\vee$B	A$\rightarrow$B	A$\leftrightarrow$B
1	1	1	1	1	1
1	0	0	1	0	0
0	1	0	1	1	0
0	0	0	0	1	1

Tabuľka 2: Pravdivostná tabuľka pre konjunkciu, disjunkciu, implikáciu a ekvivalenciu

Pomocou spomínaných krokov sa postupne dostaneme k výsledku, teda zistíme, či je konkrétna formula v konkrétnej interpretácii pravdivá alebo nepravdivá. Keď vyskúšame všetky možné kombinácie priradenia pravdivostných hodnôt výrokovým premenným a získame všetky výsledky interpretácií, môžeme podľa nich formulu zaradiť do jednej z troch skupín:

- *tautológie*: formule, ktoré súpravdivé v každej svojej interpretácii, $\top$ - symbol označujúci ľubovoľnú tautológiu (pr. $p \vee \neg p$, $p \rightarrow p$)
- *splniteľné formule*: formule, ktoré sú pravdivé aspoň v jednej svojej interpretácii (pr. $p \wedge q$, $p \vee q$)
- *kontradikcie*: formule, ktoré nie sú pravdivé v žiadnej svojej interpretácii, $\bot$ - symbol označujúci ľubovoľnú kontradikciu (pr. $p \wedge \neg p$, $p \leftrightarrow \neg p$)

4 Predikátová logika

Predikátová logika je zovšeobecnením a rozšírením výrokovej logiky. Vo výrokovej logike je elementárna formula reprezentovaná výrokovou premennou, ktorá môže nadobúdať len pravdivostné hodnoty pravda alebo nepravda. Obsah premennej pre ňu nie je podstatný, a teda ho nijak do formule nezaznamenáva. Preto, narozdiel od predikátovej logiky, nie je schopná správne určovať pravidivosť a formalizovať typy tvrdení, pre ktoré je ich význam v takomto ohľade kľúčový.

Vo všeobecnosti sú problémovými tvrdenia hovoriace o vlastnostiach alebo vzťahoch nejakých predmetov, ľudí, čísel, ich vlastností, atď. Konkrétne sú nimi výroky ako napr. „Existuje prirodzené číslo deliteľné 2 a zároveň 3.“ alebo „Pre všetky prirodzené čísla platí, že sú väčšie ako 0.“. Tieto výroky sú vo výrokovej logike atomickými výrokmi, ktorým je možné priradiť pravdivostnú hodnotu len ako celku, keďže sú reprezentované výrokovými premennými bez vnútornej štruktúry.

Predikátova logika zavádza nové prvky, ktoré jej umožňujú s takýmito typmi výrokov správne pracovať. Týmito prvkami sú *predikát* a *kvantifikátor*, a budú definované v nasledujúcich podkapitolách 4.1 a 4.2.

4.1 Formálny jazyk predikátovej logiky

Formálny jazyk predikátovej logiky, rovnako ako jazyk výrokovej logiky, označuje množinu slov takú, že každé jej slovo je formulou. Avšak, neoznačuje tú istú množinu. Abeceda formálneho jazyka predikátovej logiky je, oproti výrokovej logike, rozšírená o niekoľko ďalších symbolov, ktorými môžu byť formule tvorené. Taktiež gramatika obsahuje viac pravidiel a zavádza nový pojem - *term*.

V tejto diplomovej práci sa budem zaoberať len predikátovou logikou 1. stupňa, keďže predikátová logika 2. a vyššieho stupňa nie je predmetom vyúkového serveru, ktorý bol v rámci tejto práce vyvíjaný. Preto každé ďalšie označenie „predikátová logika“, a taktiež aj jazyk opísaný nasledujúcou abecedou a gramatikou, sa vzťahuje len na predikátovú logiku 1. stupňa, a nie vyššieho.

Abeceda formálneho jazyka predikátovej logiky obsahuje symboly, ktoré je možné rozdeliť do niekoľkých skupín:

- kvantifikátorové symboly:
  - $\forall$ - všeobecný/univerzálny kvantifikátor („pre všetky“)
  - $\exists$ - existenčný kvantifikátor(„existuje“)
- symboly logické spojky: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- predikátové symboly: zvyčajne veľké písmená abecedy, prípadne s indexmi (pr. P, Q, R)

- symboly pre indivíduové premenné: zvyčajne malé písmená abecedy, prípadne s indexmi (pr. x, y, z)
- funkčné symboly: zvyčajne malé písmená abecedy, prípadne s indexmi (pr. f, g, h)
- symboly pre indivíduové konštanty: zvyčajne malé písmená abecedy (pr. c, d, e)
- pomocné symboly: zátvorky ( ), prípadne [ ]

K predikátovým a funkčným symbolom sa udáva ešte číslo n, ktoré označuje ich aritu. N-árny symbol znamená, že symbol má aritu n. Arita je určená podľa počtu argumentov, ktoré konrétny predikát alebo funkcie obsahuje. Nulárnym predikátom je vlastne výroková premenná vo výrokovej logike a nulárnou funkciou je konštanta.

Gramatika udáva pravidlá, ktorými sú v predikátovej logike rekurzívne definované nasledujúce tri pojmy [5]:

1. termy:
   - každá indivíduová premenná je term
   - každá indivíduová konštanta je term
   - ak $x_1, \ldots, x_n$ sú termy a f je n-árna funkcia, tak aj $f(x_1, \ldots, x_n)$ je term
2. atomické formule:
   - ak $x_1, \ldots, x_n$ sú termy a P je n-árny predikátový symbol, tak $P(x_1, \ldots, x_n)$ je atomická formula
3. formule:
   - každá atomická formula je formula
   - ak A je formula, tak aj (A) je formula
   - ak A je formula, tak aj $\neg A$ je formula
   - ak A a B sú formule, tak aj $A \wedge B$, $A \vee B$, $A \rightarrow B$, $A \leftrightarrow B$ sú formule
   - ak A je formula a x je premenná, tak aj $\forall x A$ a $\exists x A$ sú formule

4.1.1 Prevod výroku z prirodzeného jazyka do jazyka predikátovej logiky

Zoberme si ako príklad tvrdenie „Pre každé reálne číslo, existuje číslo také, ktoré je od neho väčšie, a zároveň, existuje aj číslo také, ktoré je od neho menšie.“. Na tomto tvrdení si ukážeme ako funguje prevod výroku z prirodzeného jazyka do formule zapísanej vo formálnom jazyku predikátovej logiky. Ako prvé je potrebné identifikovať predikáty, konštanty a funkcie. V niektorých prípadoch, existuje viac možností akými je možné to urobiť. V našom príklade môžeme identifikovať tri atomické formule, pre ktoré zvolíme nasledujúce predikáty:

- $R(x)$: x je reálne číslo
- $V(x, y)$: x je väčšie ako y
- $M(x, y)$: x je menšie ako y

Žiadne konštanty ani funkcie sa v príklade nevyskytujú, teda ich definovanie preskočíme. V ďalšom kroku zameníme slovné spojenia „pre všetky“, prípadne „pre každé“, a „existuje“ za odpovedajúce kvantifikátorové symboly a spojenia „nie je pravda, že“, „a“, „alebo“, „ak. . . ,tak“, „práve vtedy, keď“ za odpovedajúce logické spojky. Niektoré kvantifikátory a logické spojky nie sú zadané presne týmito spojeniami, a preto ich musíme vyvodiť z kontextu.

Nakoniec, pospájaním symbolov do reťazca podľa pravidiel gramatiky s možným alebo nutným využitím zátvoriek, vznikne slovo, ktoré je formulou predikátovej logiky. Z výroku, ktorý sme si na začiatku sekcie určili ako príklad, týmto postupom dostáneme nasledujúcu formulu: $\forall x(R(x) \rightarrow (\exists y V(y, x) \wedge \exists z M(z, x)))$. V prirodzenom jazyku by sme rovnakú formulu s totožným významom predikátov mohli získať aj z podobného tvrdenia formulovaného trochu inak, a to „Ak x je reálne číslo, tak existuje číslo, ktoré je od neho väčšie, a zároveň, existuje aj číslo, ktoré je od neho menšie“.

4.2 Sémantika jazyka predikátovej logiky

Sémantika formúl v predikátovej logike je taktiež daná ich interpretáciou. Zadefinovanie interpretácií však nie je také priamočiare, ako to je pri výrokovej logike. Aby sme boli schopní formulu interpretovať, je potrebné hneď na začiatku definovať niekoľko pojmov [6].

4.2.1 Univerzum

Ako prvý pojem zadefinujeme univerzum diskurzu, nazývané aj oblasť záujmu. Univerzum je neprázdna množina hodnôt, ktoré môžu nadobúdať indivíduové premenné a konštanty nachádzajúce sa v konkrétnej formule predikátovej logiky. Stanovenie univerza je kľúčové k tomu, aby bolo možné interpretovať danú formulu.

4.2.2 Predikáty

Predikáty predstavujú elementárne formuly predikátovej logiky. Narozdiel od výrokových premenných, dokážu reprezentovať obsah a vnútornú štruktúru formúl. Sú toho schopné vďaka svojim argumentom a vďaka reláciám, ktoré definujú. Argumentmi predikátov sú termy, konkrétne indivíduové premenné, funkcie alebo konštanty. Každý predikát predstavuje nejakú reláciu, teda vzťah medzi svojimi argumentmi alebo ich vlastnosť.

Podľa počtu argumentov predikátu, sa definuje jeho arita. Takže predikát P(f(x)) je unárny predikát, teda má aritu 1, a predikát P(x,f(g(y))), čiže má aritu 2. Vo všeobecnosti sa v predikátovej logike pracuje najmä s unárnymi a binárnymi a občas s ternárnymi predikátmi. Predikáty s väčším počtom argumentov sa používajú zriedkavo.

4.2.3 Kvantifikátory

V predikátovej logike existujú dva typy kvantifikátorov. Všeobecný kvantifikátor, ktorý je možné do prirodzeného jazyka preložiť ako „pre všetky prvky z daného univerza“ a existenčný kvantifikátor, ktorý môže byť preložený ako „existuje prvok z daného univerza“. Taktiež môžu vo formule existovať aj predikáty, na ktoré sa nevzťahuje žiadny kvantifikátor. Jeden kvantifikátor sa môže vzťahovať od jedného predikátu až na nekonečne veľa. To, či sa daný kvantifikátor na nejaký predikát vzťahuje, je určené tým, že predikát obsahuje premennú s rovnakým menom aké má aj premenná pri kvantifikátore. Ak je v nejakej formule viac kvantifikátorov s prekrývajúcou sa pôsobnosťou, tak sa aplikujú postupne od najvnútornejšieho k najvonkajšiemu.

4.2.4 Indivíduové premenné

Indivíduové premenné reprezentujú vnútorné objekty atomických formúl, ktoré môžu nadobúdať ľubovoľné hodnoty z daného univerza. Indivíduové premenné je možné rozdeliť do dvoch skupín v závislosti na tom, či sa na ne vzťahuje nejaký kvantifikátor:

- viazané: majú viazaný výskyt vo formule, sú kvantifikované
- voľné: majú voľný výskyt vo formule, nie sú kvantifikované

Premenná x je kvantifikovaná práve vtedy, keď je súčasťou nejakej podformule tvaru $\forall x P(x)$ alebo $\exists x P(x)$. V niektorých formulách môže mať jedna premenná voľný, a zároveň aj viazaný výskyt, napr. premenná x vo formule $P(x) \wedge \forall x \exists y R(x, y)$. V takýchto prípadoch je dobré, často aj potrebné, jeden výskyt premennej premenovať, a to hlavne v prípade vykonávania ekvivalentných úprav.

Podľa výskytu indivíduových premenných sa formule delia na dve skupiny. Ak sa vo formule nachádza aspoň jedna voľná premenná, tak sa formula nazýva otvorená. Ak formula neobsahuje žiadnu voľnú premennú, tak je uzavretá.

4.2.5 Interpretovanie formúl predikátovej logiky

Zadefinovanie konkrétnej interpretácie pre nejakú formulu, je možné zhrnúť do nasledujúcich krokov:

1. zadefinovanie univerza U (U^n definuje kartézsky súčin, tj. $Ux \dots xU$)
2. priradenie určitej n-árnej relácie každému n-árnemu predikátovému symbolu (n-árnu reláciu je možné definovať aj ako zobrazenie $Un \to \{0, 1\}$)
3. priradenie určitého zobrazenia $f : U^n \to U$ každému n-árnemu funkčnému symbolu (každý n-árny funkčný symbol je interpretovaný ako úplná funkcia, ktorá n-tici indivíduí priradí jedno indivíduum)
4. priradenie jednej ľubovoľnej hodnoty z univerza každej konštante

V prípade voľných premenných je potrebné použiť valuáciu, nazývanú aj ohodnotenie indivíduových premenných. Valuácia spočíva v priradení jednej ľubovoľnej hodnoty z univerza každej nekvantifikovanej premennej.

Ukážkové zadanie interpretácie nejakej formule môže vyzerať takto:

1. univerzum: $1, 2, 3$
2. relácie predikátových symbolov:
   - relácia unárneho predikátu $P : \{(1), (2)\}$
   - relácia binárneho predikátu $R : \{(1, 3), (2, 1), (3, 2)\}$
3. zobrazenia funkčných symbolov:
   - zobrazenie unárnej funkcie $f : \{1 \to 2, 2 \to 3, 3 \to 1\}$
   - zobrazenie binárnej funkcie $g : \{(1, 1) \to 2, (1, 2) \to 3, (1, 3) \to 1, (2, 1) \to 2, (2, 2) \to 3, (2, 3) \to 1, (3, 1) \to 2, (3, 2) \to 3, (3, 3) \to 1, \}$
4. interpretácia konštanty $c : 2$
5. valuácia voľnej premennej $y : 1$

Každá interpretácia ľubovoľnej formule jazyka predikátovej logiky môže byť, buď pravdivá, alebo nepravdivá. Interpretácia, v ktorej je formula pravdivá, sa nazýva modelom tejto formule. Podľa výsledkov jednotlivých interpretácií, taktiež ako vo výrokovej logike, sa formule predikátovej logiky rozdeľujú do troch skupín:

- tautológie (logicky pravdivé formule): formule, ktoré sú pravdivé v každej svojej interpretácii, teda každá interpretácia je ich modelom; označuje sa $\models A$ pre formulu A

- splniteľné formule: formule, ktoré sú pravdivé aspoň v jednej interpretácii, teda pre ne existuje nejaká interpretácia, ktorá je modelom
- kontradikcie: formule, ktoré nie sú pravdivé v žiadnej interpretácii, teda neexistuje pre ne ani jeden model

5 Ekvivalentné úpravy

Táto kapitola sa zaoberá *ekvivalentnými úpravami* formúl výrokovej a predikátovej logiky. Konkrétne prezentuje a zaraďuje do skupín logické zákony fungujúce na princípe ekvivalencie medzi formulami [7].

Pod ekvivalentnými úpravami formúl sa rozumejú úpravy, ktorými je možné previesť formulu výrokovej alebo predikátovej logiky na formulu v inom tvare, ktorá je s ňou *logicky ekvivalentná*. Dve formule A a B sú logicky ekvivalentné práve vtedy, keď formula $A \leftrightarrow B$ je tautológia. Inými slovami, tieto formule majú rovnakú pravdivostnú hodnotu, keď sú interpretované rovnakým spôsobom a navzájom zo seba vyplývajú.

Pomocou ekvivalentných úprav je možné formule prevádzať do špeciálnych ekvivalentných tvarov, ako sú napr. *konjunktívna* alebo *disjunktívna normálna forma*. Tieto a ďalšie špeciálne formy budú viac opísané v kapitole 6. Označenia A, B a C v nasledujúcich podkapitolách reprezentujú formuly. Buď všetky reprezentujú ľubovoľné formule výrokovej logiky alebo všetky reprezentujú ľubovoľné formule predikátovej logiky, dokým nie je uvedené inak v konkrétnom prípade.

5.1 Základné zákony reprezentujúce ekvivalentné úpravy

Nasledujúce zákony reprezentujú najzákladnejšie ekvivalentné úpravy používané pri práci s formulami výrokovej a predikátovej logiky:

- identita: $A \Leftrightarrow A$
- dvojitá negácia: $\neg\neg A \Leftrightarrow A$
- idempotencia konjunkcie: $A \wedge A \Leftrightarrow A$
- idempotencia disjunkcie: $A \vee A \Leftrightarrow A$
- absorpcia disjunkcie: $A \wedge (A \vee B) \Leftrightarrow A$
- absorpcia konjunkcie: $A \vee (A \wedge B) \Leftrightarrow A$
- konjunkcia s tautológiou: $A \wedge \top \Leftrightarrow A$
- disjunkcia s kontradikciou: $A \vee \bot \Leftrightarrow A$
- disjunkcia s tautológiou: $A \vee \top \Leftrightarrow \top$
- konjunkcia s kontradikciou: $A \wedge \bot \Leftrightarrow \bot$

5.2 Komutatívne zákony

Komutatívnosť je vlastnosť operácií, ktorá predstavuje nezávislosť poradia operandov pre nejakú operáciu [8]. Operandy sú vstupnými hodnotami operácií. Inými slovami, ak je operácia komutatívna, tak výsledok po jej aplikovaní vyjde rovnaký pre akékoľvek poradie operandov. Operácie, ktoré nie sú komutatívne, sa nazývajú nekomutatívne.

Vo výrokovej aj predikátovej logike je jedinou nekomutatívnou binárnou operáciou implikácia. To znamená, že komutatívne zákony sú definované pre ostatné binárne logické operátory, a to pre konjunkciu, disjunkciu a ekvivalenciu. Tieto zákony hovoria o tom, že daná formula je ekvivalentá pred a po prehodení vstupov komutatívnych logických operácií. Sú nimi konkrétne:

- konjunkcia: $A \wedge B \Leftrightarrow B \wedge A$
- disjunkcia: $A \vee B \Leftrightarrow B \vee A$
- ekvivalencia: $A \leftrightarrow B \Leftrightarrow B \leftrightarrow A$

Okrem komutatívnych zákonov pre operátory sa v predikátovej logike navyše definujú aj komutatívne zákony pre kvantifikátory:

- 2 všeobecné kvantifikátory: $\forall x \forall y P(x, y) \Leftrightarrow \forall y \forall x P(x, y)$
- 2 existenčné kvantifikátory: $\exists x \exists y P(x, y) \Leftrightarrow \exists y \exists x P(x, y)$
- 1 všeobecný a 1 existenčný kvantifikátor: $\exists x \forall y P(x, y) \Rightarrow \forall y \exists x P(x, y)$

V prípade, že sa vo formule bezprostredne za sebou nachádzajú kvantifikátory rôzneho typu, ako je to v zákone popísanom v poslednej odrážke, tak pre ne komutatívnosť platí len pri prevode jedným smerom. Na tento fakt je potrebné si dávať pozor pri prevode formúl do ekvivalentých tvarov.

5.3 Asociatívne zákony

Asociatívnosť je vlastnosť operácií, ktorá hovorí o tom, že keď výraz obsahuje viacero výskytov tej istej operácie za sebou, nezáleží na uzátvorkovaní tejto časti a na poradí vyhodnocovania operátorov [9]. Operácie spĺňajúce túto vlastnosť sa nazývajú asociatívne. Ostatné, ktoré ju nespĺňajú, sú neasociatívne.

Vo výrokovej aj predikátovej logike sú asociatívne zákony definované pre logické operácie konjunkciu, disjunkciu a ekvivalenciu. To znamená, že formule, ktoré obsahujú len jeden z týchto troch operátov v akomkoľvek množstve, sú logicky ekvivalentné pre akékoľvek uzátvorkovanie a poradie vyhodnocovania operátorov. Asociatívne zákony sú definované nasledovne pomocou ekvivalencií:

- konjunkcia: $(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$
- disjunkcia: $(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$
- ekvivalencia: $(A \leftrightarrow B) \leftrightarrow C \Leftrightarrow A \leftrightarrow (B \leftrightarrow C)$

5.4 Distributívne zákony

Distributívnosť je vlastnosť spočívajúca v možnosti distribúcie jednej operácie cez inú operáciu [10]. Operácia, ktorú je možné distribuovať cez nejakú inú operáciu o na určitej množine, sa nazýva distributívna voči operácii o na danej množine.

Vo výrokovej a predikátovej logike je možné distribuovať logické operácie konjunkciu a disjunkciu navzájom cez seba. Konjunkcia je distributívna voči disjunkcii, a naopak disjunkcia je distributívna voči konjunkcii. Zákony, ktoré vyjadrujú tieto vzťahy sú definované pomocou nasledujúcich ekvivalencií:

- $(A \wedge B) \vee C \Leftrightarrow (A \vee C) \wedge (B \vee C)$
- $(A \vee B) \wedge C \Leftrightarrow (A \wedge C) \vee (B \wedge C)$

Z nich vyplýva, že formula je logicky ekvivalentná s formulou, ktorá vznikne distribúciou operácie konjunkcie cez disjunkciu, alebo naopak.

Okrem distributívnych zákonov pre operátory, sa v predikátovej logike ďalej uvádzajú aj distributívne zákony pre kvantifikátory. Tieto zákony vyjadrujú distribúciu kvantifikátorov cez logické operácie konjunkciu, disjunkciu a implikáciu [11]. Na základe sily vyplývania ich môžeme rozdeliť do dvoch skupín:

1. distributívne zákony, ktoré vyjadrujú ekvivalenciu medzi formulami, tj. formule zo seba navzájom vyplývajú
   - $\forall x(P(x) \wedge R(x)) \Leftrightarrow (\forall x P(x) \wedge \forall x R(x))$
   - $\exists x(P(x) \vee R(x)) \Leftrightarrow (\exists x P(x) \vee \exists x R(x))$
   - $\exists x(P(x) \rightarrow R(x)) \Leftrightarrow (\forall x P(x) \rightarrow \exists x R(x))$
2. distributívne zákony, ktoré vyjadrujú implikáciu medzi formulami, tj. vyplývanie funguje len jedným smerom
   - $\exists x(P(x) \wedge R(x)) \Rightarrow (\exists x P(x) \wedge \exists x R(x))$
   - $(\forall x P(x) \vee \forall x R(x)) \Rightarrow \forall x(P(x) \vee R(x))$
   - $(\exists x P(x) \rightarrow \forall x R(x)) \Rightarrow \forall x(P(x) \rightarrow R(x))$

5.5 De Morganove zákony

V teórii množín, *De Morganove zákony* vyjadrujú vzťah medzi prienikom $\cap$ a zjednotením $\cup$ za pomoci doplnkov M'. Definujú ako prevávadzať medzi týmito množinovými operáciami. Pre ľubovoľné množiny M a N sú definované ako tieto rovnosti [12]:

- $(M \cap N)' = (M' \cup N')$
- $(M \cup N)' = (M' \cap N')$

Vo výrokovej a predikátovej logike je možné tento istý vzťah vyjadriť cez logické operácie konjunkciu, disjunkciu a negáciu. Konjunkcia reprezentuje prienik, disjunkcia zjednotenie a negácia doplnok. De Morganove zákony v rámci logiky, teda vyjadrujú vzťah medzi konjunkciou a disjunkciou za pomoci negácie. Vo formálnom jazyku sú tieto zákony zapísané ako nasledovné ekvivalencie:

- negácia konjunkcie: $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$
- negácia disjunkcie: $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$

Pre predikátovú logiku existujú aj ďalšie verzie De Morganových zákonov, a to pre kvantifikátory. Ich základnou myšlienkou je to, že aplikovanie negácie na kvantifikátor ho zmení na opačný. Teda existenčný kvantifikátor sa zmení na všeobecný, a naopak. De Morganove zákony pre kvantifikátory vyzerajú nasledovne [13]:

- $\neg\forall x P(x) \Leftrightarrow \exists x \neg P(x)$
- $\neg\exists x P(x) \Leftrightarrow \forall x \neg P(x)$

5.6 Zákony pre implikáciu a ekvivalenciu

Zákony v tejto podkapitole definujú vzťahy medzi implikáciou a ostatnými logickými operátormi, a taktiež ekvivalenciou a ostatnými logickými operátormi. Využívajú sa prevažne vtedy, keď je potrebné ekvivalenciu alebo implikáciu odstraňovať pre jednoduchšiu prácu s formulami výrokovej alebo predikátovej logiky. Sú definované pomocou ekvivalencií rozdelených do dvoch skupín:

1. prevody implikácie
    - negácia implikácie: $\neg(A \rightarrow B) \Leftrightarrow (A \wedge \neg B)$
    - $(A \rightarrow B) \Leftrightarrow (\neg A \vee B)$
2. prevody ekvivalencie
    - negácia ekvivalencie: $\neg(A \leftrightarrow B) \Leftrightarrow (A \wedge \neg B) \vee (\neg A \wedge B)$
    - $(A \leftrightarrow B) \Leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$
    - $(A \leftrightarrow B) \Leftrightarrow (A \wedge B) \vee (\neg B \wedge \neg A)$

5.7 Zákony pre prevod do prenexového tvaru

Prenexový tvar formúl je špeciálnym ekvivalentným tvarom formúl predikátovej logiky takým, že všetky kvantifikátory sa nachádzajú v tzv. prefixe, teda na začiatku formule. Časť formule za prefixom už žiadny kvantifikátor neobsahuje. Viac do hĺbky sa budem prenexovou formou zaoberať v podkapitole 6.3.

Zákony v tejto podkapitole definujú možnosti presunutia kvantifikátorov na začiatok formule, teda možnosti prevod formule do prenexového tvaru. Rozdelíme ich do troch skupín podľa logického operátora [14]:

1. konjunkcia
   - $\forall x(A \wedge B(x)) \Leftrightarrow (A \wedge \forall x B(x))$
   - $\exists x(A \wedge B(x)) \Leftrightarrow (A \wedge \exists x B(x))$
2. disjunkcia
   - $\forall x(A \vee B(x)) \Leftrightarrow (A \vee \forall x B(x))$
   - $\exists x(A \vee B(x)) \Leftrightarrow (A \wedge \exists x B(x))$
3. implikácia
   - $\forall x(A \rightarrow B(x)) \Leftrightarrow (\exists x B(x) \rightarrow A)$
   - $\exists x(A \rightarrow B(x)) \Leftrightarrow (\forall x B(x) \rightarrow A)$
   - $\forall x(A \rightarrow B(x)) \Leftrightarrow (A \rightarrow \forall x B(x))$
   - $\exists x(A \rightarrow B(x)) \Leftrightarrow (A \rightarrow \exists x B(x))$

V uvedených zákonoch označenie A reprezentuje akúkoľvek formulu predikátovej logiky, v ktorej sa nevyskytuje voľná premenná a označenie $B(x)$ akúkoľvek formulu predikátovej logiky, v ktorej sa premenná x vyskytovať môže.

6 Špeciálne tvary formúl

Špeciálne tvary formúl sú vo väčšine ekvivalentné tvary, do ktorých je možné formule výrokovej alebo predikátovej logiky prevádzať. Tieto tvary sa využívajú za rôznymi účelmi, napr. sprehľadnenie zápisu či zjednodušenie určovania pravdivosti. Jednými z najpoužívanejších špeciálnych tvarov sú konjunktívna a disjunktívna normálna forma pre formule výrokovej logiky, a prenexový tvar pre formule predikátovej logiky.

6.1 Konjunktívna normálna forma

Konjunktívna normálna forma formule, skrátene KNF, je ekvivalentný tvar formule, v ktorom sa nachádzajú z logických spojok len spojky negácia, konjunkcia a disjunkcia, a to ešte v špecifickom tvare. Tento konkrétny tvar, aký majú formule mať, aby boli V KNF, je opísaný v nasledujúcich bodoch [15]:

- literál: výrokový symbol alebo jeho negácia (pr. $p, \neg p$)
- disjunktívna klauzula: disjunkcia literálov(pr. $p \vee q$)
- KNF: konjunkcia klauzulí (pr. $(p \vee \neg q) \wedge (\neg p \vee q)$)

Postup pre prevod ľubovoľnej formule výrokovej logiky do konjunktívnej normálnej formy sa skladá z piatich hlavných krokov:

1. odstránenie ekvivalencií pomocou zákonov definovaných v podkapitole 5.6
2. odstránenie implikácií pomocou zákonov definovaných v podkapitole 5.6
3. použitie De Morganových zákonov definovaných v podkapitole 5.5
4. použitie distributívnych zákonov definovaných v podkapitole 5.4
5. použitie asociatívnych zákonov definovaných v podkapitole 5.3

Ďalším možným krokom je odstránenie tautológii tvaru $(p \vee \neg p)$, čo sa používa pre zjednodušenie vzniknutej formuly, definované v podkapitole 5.1.

Okrem základnej KNF, existuje ešte aj špeciálny ekvivalentný tvar KNF nazývaný *úplná konjuktívna normálna forma* (ÚKNF). Každú formulu, ktorá nie je tautológiou, je možné previesť do tvaru ÚKNF.

Aby formula bola v ÚKNF, jej klauzule musia byť úplnými disjunktívnymi klauzulami. Úplné disjunktívne klauzule sú špeciálne klauzule, ktorých literály sú spojené disjunkciou a obsahujú každý výrokový symbol, alebo jeho negáciu, práve raz. Príkladom formule, ktorá sa nachádza v tvare ÚKNF je $(p \vee \neg q \vee r \vee \neg s) \wedge (\neg p \vee q \vee r \vee s)$ v prípade, že množina výrokových symbolov je $\{p, q, r, s\}$.

6.1.1 Príklad prevodu formuly do KNF

Ako príklad si zoberme formulu $\neg(\neg p \vee q) \wedge (p \leftrightarrow \neg q)$. Aby sme ju previedli do KNF, budeme nasledovať kroky z predchádzajúcej podkapitoly. Konkrétny postup je rozpísaný v bodoch pre lepšiu prehľadnosť.

1. odstránením ekvivalencie použitím zákonu $(A \leftrightarrow B) \Leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$ vznikne formula $\neg(\neg p \vee q) \wedge ((p \rightarrow \neg q) \wedge (\neg q \rightarrow p))$
2. odstránením prvej implikácie použitím zákonu $(A \rightarrow B) \Leftrightarrow (\neg A \vee B)$ vznikne formula $\neg(\neg p \vee q) \wedge ((\neg p \vee \neg q) \wedge (\neg q \rightarrow p))$
3. odstránením druhej implikácie použitím rovnakého zákonu vznikne formula $\neg(\neg p \vee q) \wedge ((\neg p \vee \neg q) \wedge (q \vee p))$
4. použitím De Morganovho zákonu $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$ vznikne formula $(p \wedge \neg q) \wedge ((\neg p \vee \neg q) \wedge (q \vee p))$
5. použitie distributívnych zákonov nie je v tomto prípade potrebné, preto tento krok môžeme preskočiť
6. použitím asociatívnych zákonov sa odstránia nadbytočné zátvorky, a tým získame výslednú formulu v KNF $p \wedge \neg q \wedge (\neg p \vee \neg q) \wedge (q \vee p)$

6.2 Disjunktívna normálna forma

Disjunktívna normálna forma formule, skrátene DNF, je ekvivalentný tvar danej formule, v ktorom sa, rovnako ako v konjunktívnej normálnej forme, nachádzajú z logických spojok len spojky negácia, konjunkcia a disjunkcia v špecifickom tvare. Tvar formúl v DNF, ale nie je rovnaký s tvarom formúl KNF, a je opísaný v nasledujúcich bodoch [16]:

- literál: výrokový alebo predikátový symbol, alebo jeho negácia
- konjunktívna klauzula: konjunkcia literálov (pr. $p \wedge q$)
- DNF: disjunkcia klauzulí (pr. $(p \wedge \neg q) \vee (\neg p \wedge q)$)

Hlavný postup pre prevod ľubovoľnej formule výrokovej logiky do disjunktívnej normálnej formy je zložený z rovnakých krokov ako prevod formule do KNF s tým rozdielom, že zákony používame s iným cieľom.

Pre zjednodušenie vzniknutej formuly je ďalej možné v určitých prípadoch využiť ekvivalentnú úpravu odstránenia kontradikcií v tvare $(p \wedge \neg p)$, ktorá je zadefinovaná v podkapitole 5.1.

Okrem základnej DNF, podobne ako pri KNF, existuje ešte aj špeciálna forma DNF. Je nazývaná *úplná disjuktívna normálna forma* (ÚKNF). Každú formulu, ktorá nie je kontradikciou, je možné previesť do tvaru ÚDNF.

Aby nejaká formula bola v ÚDNF, jej klauzule musia byť úplnými konjunktívnymi klauzulami. Úplné konjunktívne klauzule sú špeciálne klauzule, ktorých literály sú spojené konjunkciou a obsahujú každý výrokový symbol, alebo jeho negáciu, práve raz. Príkladom formule v tvare ÚDNF je $(\neg p \wedge q \wedge \neg r \wedge s) \vee (p \wedge q \wedge \neg r \wedge \neg s)$ v prípade, že množina výrokových symbolov je $\{p, q, r, s\}$.

6.2.1 Príklad prevodu formuly do DNF

Ako príklad si zoberme formulu $\neg(\neg(p \rightarrow q) \vee q)$. Aby sme ju previedli do DNF, budeme nasledovať rovnaké kroky ako pre prevod do KNF. Konkrétne kroky sú znovu rozpísané v bodoch pre lepšiu prehľadnosť.

1. v ukážkovej formule sa žiadna ekvivalencia nenachádza, takže tento krok preskočíme
2. odstránením implikácie použitím zákonu $\neg(A \rightarrow B) \Leftrightarrow (A \wedge \neg B)$ vznikne formula $\neg((p \wedge \neg q) \vee q)$
3. použitím De Morganovho zákonu negácie disunkcie $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$ vznikne formula $(\neg(p \wedge \neg q) \wedge \neg q)$
4. použitím De Morganovho zákonu negácie konjunkcie $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$ vznikne formula $((\neg p \vee q) \wedge \neg q)$
5. použitím distributívneho zákonu $(A \vee B) \wedge C \Leftrightarrow (A \wedge C) \vee (B \wedge C)$ vznikne formula $((\neg p \wedge \neg q) \vee (q \wedge \neg q))$
6. pomocou asociatívnych zákonov môžeme odstrániť vonkajšiu zátvorku, a tým získať formulu v tvare $(\neg p \wedge \neg q) \vee (q \wedge \neg q)$
7. formula v predchádzajúcom bode sa už v DNF nachádza, ale môžeme ju ešte ďalej zjednodušiť odstránením kontradikcie $(q \wedge \neg q)$; výsledná formulu v DNF teda bude vyzerať $(\neg p \wedge \neg q)$

6.3 Prenexová normálna forma

Prenexový tvar formúl predikátovej logiky je ekvivalentý tvar, v ktorom sa všetky kvantifikátory nachádzajú v prefixe formule, a teda zvyšok formule neobsahuje žiadny kvantifikátor. Každú formula predikátovej je možné previesť do tohto tvaru. Príkladom formule v prenexovom tvare je $\forall x \exists y \forall z(P(x, y) \rightarrow P(x, z))$.

Prenexová normálna forma, skrátene PNF, je špeciálny prenexový tvar, v ktorom je časť za kvantifikátormi v konjuktívnej alebo disjunktívnej normálnej forme. Príkladom formule v PNF je $\forall x \exists y \exists w((P(x, g(y)) \wedge R(z, w)) \vee ((R(w, f(x)) \wedge P(c, y)))$. Postup prevodu formule predikátovej logiky do PNF vyzerá nasledovne [17]:

1. odstránenie ekvivalencií pomocou zákonov definovaných v podkapitole 5.6
2. odstránenie implikácií pomocou zákonov definovaných v podkapitole 5.6
3. použitie De Morganových zákonov definovaných v podkapitole 5.5
4. premenovanie viazaných a voľných indivíduových premenných tak, aby sa žiadne dve z nich nevolali rovnako
5. odstránenie nadbytočných kvantifikátorov
6. použitie zákonov pre prevod formule do prenexového tvaru definovaných v podkapitole 5.7
7. použitie distributívnych zákonov definovaných v podkapitole 5.4
8. použitie asociatívnych zákonov definovaných v podkapitole 5.3

6.3.1 Príklad prevodu formuly do PNF

Pre ukázanie prevodu formuly predikátovej logiky do PNF si zoberme formulu $\forall x(\exists y P(x, y) \wedge \forall y(P(y, x) \rightarrow R(z)))$. Postup pre prevod tejto konkrétnej formule vychádza z krokov definovaných v predchádzajúcom bode.

1. vo formule sa žiadna ekvivalencia nenachádza, takže tento krok môžeme preskočiť
2. odstránením implikácie použitím zákonu $(A \rightarrow B) \Leftrightarrow (\neg A \vee B)$ vznikne formula $\forall x(\exists y P(x, y) \wedge \forall y(\neg P(y, x) \vee R(z)))$
3. použitie De Morganových zákonov nie je v tomto prípade potrebné
4. premenovaním viazaných aj voľných indivíduových premenných získame formulu $\forall x(\exists y P(x, y) \wedge \forall w(\neg P(w, x) \vee R(z)))$
5. v ukážkovej formule sa nenachádza žiadny nadbytočný kvantifikátor, ktorý by bolo treba odstrániť
6. použitím zákonov pre prevod do prenexového tvaru z bodu 5.7 dostaneme $\forall x \exists y \forall w(P(x, y) \wedge (\neg P(w, x) \vee R(z)))$, ktorá sa už nachádza v prenexovej konjunktívnej normálnej forme
7. ak by sme chceli formulu dostať do prenexovej disjunktívnej normálnej formy, potrebujeme použiť distributívny zákon $(A \vee B) \wedge C \Leftrightarrow (A \wedge C) \vee (B \wedge C)$; výslednou formulou sa teda stanej $\forall x \exists y \forall w((P(x, y) \wedge \neg P(w, x)) \vee (P(x, y) \wedge R(z)))$

7 Webová aplikácia

Webová aplikácia, ktorá bola vyvíjaná v rámci tejto diplomovej práce, je zameraná na prácu s formulami predikátovej logiky. Je rozdelená na štyri časti, z ktorých každá umožňuje vykonávať iné operácie nad zadanými formulami. Týmito operáciami sú syntaktická kontrola formúl predikátovej logiky, zobrazenie syntaktického stromu, prevedenie formule do prenexovej normálnej formy a vyhodnotenie pravdivosti formule na základe zadaných interpretácií.

Aplikácia bola vytvorená s cieľom nasadenia na výukový server pre teoretickú informatiku, ktorý vzniká v rámci diplomových a bakalárskych prác na VŠB-TUO. Mala by slúžiť ako pomoc pri riešení a pochopení úloh z oblasti predikátovej logiky. Jej používateľské rozhranie je v českom jazyku, keďže prevažne v tomto jazyku sa na VŠB-TUO vyučujú predmety zamerané na teoretickú informatiku.

7.1 Použité technológie

Na vývoj webovej aplikácie bola použitá technológia *ASP.NET Core*, ktorá je súčasťou vývojárskej platformy *.NET Framework*. Vývoj prebiehal vo vývojovom prostredí Microsoft Visual Studio 2017, keďže ponúka dobrú podporu pre vývoj aplikácii využívajúcich technológiu .NET Framework.

.NET Framework je bezplatná otvorená vývojárska platforma od spoločnosti Microsoft, pomocou ktorej je možné vyvíjať rôzne druhy aplikácií naprieč platformami [18]. Technológia ASP.NET Core je prestavaná verzia ASP.NET frameworku a vďaka architektonickým zmenám je odľahčenejším a modulárnejším frameworkom [19]. ASP.NET Core aj ASP.NET sa využívajú na vývoj dynamických webových stránok s využitím .NET Frameworku a jazyku C#.

Konkrétne bol na vývoj webovej aplikácie použitý framework *Razor Pages*. Už z jeho názvu vyplýva, že rozdeľuje vývoj aplikácií podľa stránok (Pages). Razor stránka je súbor s príponou .cshtml obsahujúci kód v značkovacom jazyku Razor, v ktorom sa vytvára používateľské rozhranie. Každá stránka má svoj model (trieda PageModel), ktorý rieši jej funkcionalitu. Je ním súbor s príponou .cshtml.cs s rovnakým menom ako má stránka, obsahujúci kód písaný v jazyku C#.

Hneď na začiatku každej stránky v jazyku Razor sa nachádza označenie `@page`, ktoré jej umožňuje spracovávať požiadavky priamo bez použitia radiča (controller). Jazyk Razor je tvorený prevažne jazykom HTML, ale umožňuje zakomponovať aj kód napísaný v C#, a to pomocou symbolu @, viď Výpis 1, ktorý pochádza z vyvinutej webovej aplikácie. vykresľovanie jazyka Razor je rovnaké ako vykresľovanie čistého HTML.

```
 @if (Model.Message != null)
{
    <div class="text-center">
        @if (Model.Message == "Řetězec je formulí predikátové logiky")
        {
            <p class="line-spaces-and-breaks well well-ok" style="font-size:18px
                ">@Model.Message</p>
        }
        else if (Model.Message == "Řetězec je termem")
        {
            <p class="line-spaces-and-breaks well well-term" style="font-size:18
                px">@Model.Message</p>
        }
        else
        {
            <p class="line-spaces-and-breaks well well-nok" style="font-size:18
                px">@Model.Message</p>
        }
    </div>
}
```

Výpis 1: Príklad použitia jazyka Razor

Razor Pages framework je založený na ASP.NET Core MVC frameworku [20]. ASP.NET Core MVC je súčasťou ASP.NET Core technológie a je zameraný na vývoj webových aplikácií pomocou architektonického štýlu Model-View-Controller, teda skrátene MVC. Architektúra MVC rozdeľuje aplikáciu na tri prepojené časti:

- view - prezentačná vrstva
- model - doménová logika
- controller - radič

Hlavným cieľom MVC je oddelenie prezentačnej vrstvy od doménovej logiky, a to tak že prezentácia na doménovej logike závisí, ale logika na prezentácii nie. Radič (controller) je medziprvok medzi modelom a prezentáciou. Prijíma používateľské vstupy, ktoré predáva modelu, a na základe spracovaných dát vie ďalej aktualizovať prezentáciu.

Razor Pages používajú odľahčenejšiu verziu architektúry MVC, keďže kód reprezentujúci radič sa nachádza v stránke samotnej a nie je potrebné pre neho vytvárať vlastnú triedu.

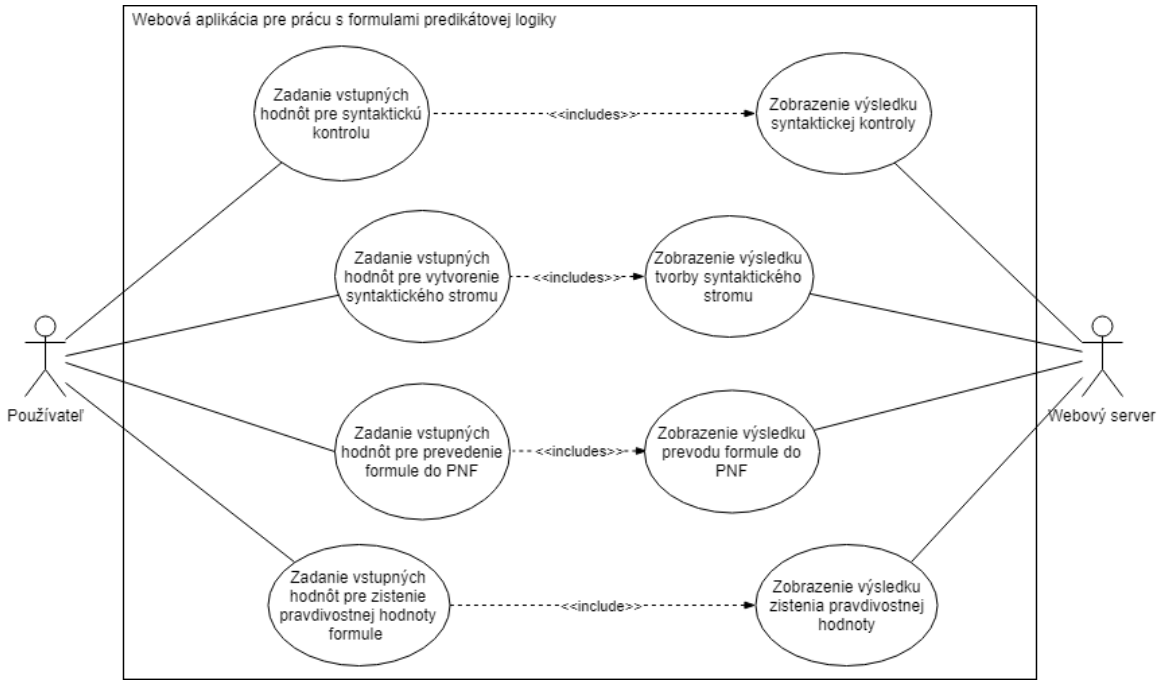
Pre testovanie aplikácie boli využívané webové prehliadače Mozilla Firefox a Chrome. Testovanie prebiehalo pri rozlíšení 1920x1080px.

7.2 Reprezentácia aplikácie pomocou diagramov

Diagramy sú štruktúrované grafické zobrazenia slúžiace k znázoreniu nejakých informácií. Pre tvorbu diagramov sa v softvérovom inžinierstve využíva modelovací jazyk UML (Unified Modeling Language). Jazyk UML bol použitý aj na vytvorenie nasledujúcich dvoch diagramov.

7.2.1 Prípad použitia

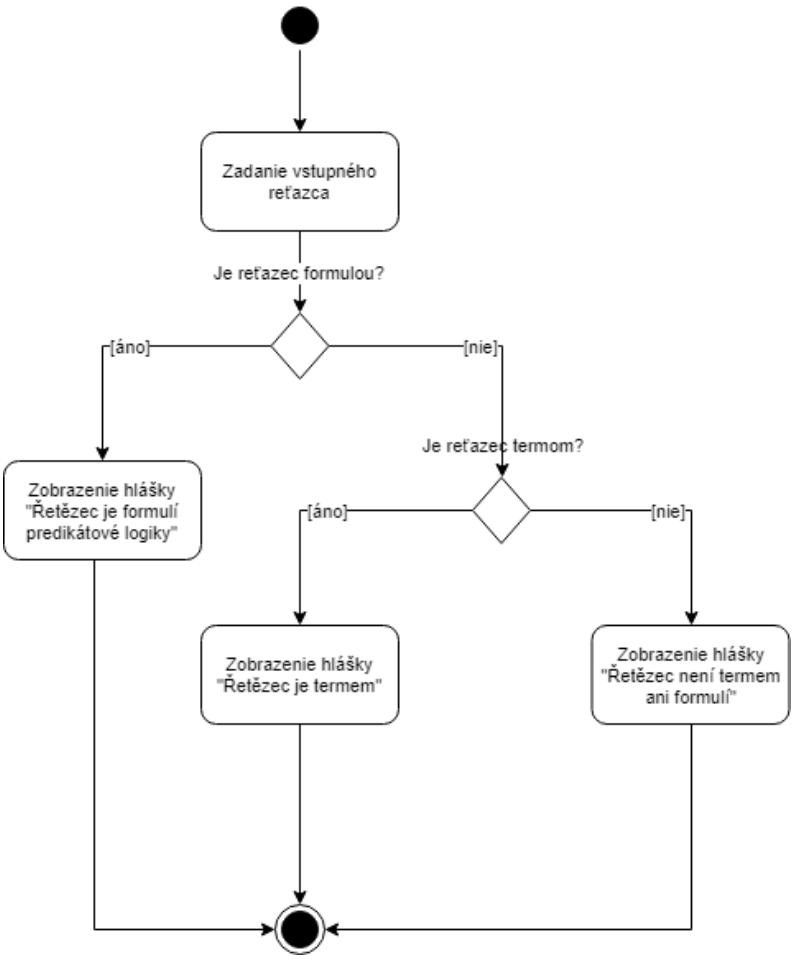
Prípad použitia, v angličtine use case, slúži na zobrazenie možností použitia systému učastníkmi systému. Prípad použitia na obrázku 1 reprezentuje základné možnosti použitia webovovej aplikácie.



Obr. 1: Prípad užitia reprezentujúci prácu s webovou aplikáciou

7.2.2 Diagram aktivít

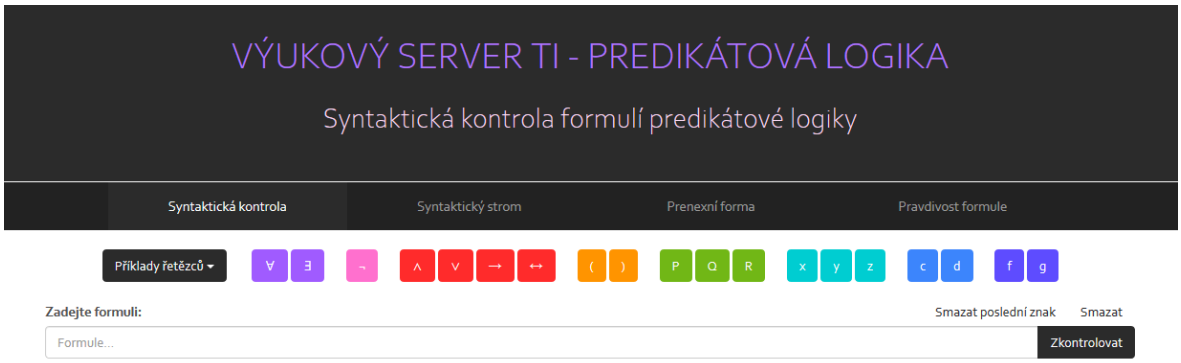
Diagram aktivít slúži na popísanie aktivít v postupnosti, v akej sa v systéme odohrávajú. Reprezentuje správanie systému v nejakom konkrétnom prípade. Diagram aktivít zobrazený na obrázku 2 predstavuje proces syntaktickej analýzy jedného reťazca.



Obr. 2: Diagram aktivít reprezentujúci vykonanie syntaktickej kontroly jedného zadaného reťazca

8 Implementácia webovej aplikácie

Vo webovej aplikácii je každej spomínanej časti z oblasti predikátovej logiky venovaná jedna stránka. Každá stránka obsahuje nadpis *VÝUKOVÝ SERVER TI - PREDIKÁTOVÁ LOGIKA*, popis toho, na čo sa zameriava, nápovedu, navigačnú lištu a vstupné pole pre zadanie reťazca s možným využitím tlačidiel a príkladov, viď Obr.3. V časti *Syntaktický strom* je ďalej možné zvoliť typ vykreslenia stromu pomocou políčka *Rozložit predikáty*, v časti *Prenexová forma* je možné zvoliť typ normálnej formy, a v časti *Pravdivost formule* sa nachádza niekoľko ďalších vstupných polí a tlačidiel na zadávanie rôznych druhov interpretácií.



Obr. 3: Ukážka základnej štruktúry stránky

8.1 Syntaktická kontrola

Prvá časť webovej aplikácie sa zaoberá syntaktickou kontrolou formúl, teda zisťuje, či je daný reťazec formulou predikátovej logiky. Syntaktická kontrola funguje na princípe syntaktickej analýzy. Aby bolo možné syntaktickú analýzu vykonať, je potrebné najprv definovať formálnu gramatiku, podľa ktorej má pracovať.

Vo väčšine prípadov je gramatika syntaktickej analýzy bezkontextovou gramatikou. Bezkontextová gramatika je formálna gramatika obsahujúca len pravidlá tvaru $X \longrightarrow x$, kde X je neterminál a x je reťazec zložený z terminálov (symbolov abecedy daného formálneho jazyka) a neterminálov.

Gramatika, na základe ktorej je postavená syntaktická kontrola vo webovej aplikácii, je taktiež bezkontextovou gramatikou. Je tvorená množinou dvanástich neterminálov, abecedou, počiatočným neterminálom S a pravidlami.

Abeceda je upravenou verziou abecedy formálneho jazyka predikátovej logiky, viď podkapitola 4.1, a vyzerá nasledovne:

- kvantifikátorové symboly:
  - . všeobecný kvantifikátor - $\forall$
  - . existenčný kvantifikátor - $\exists$
- symboly logické spojky:
  - . negácia - $\neg$, !
  - . konjunkcia - $\wedge$, &
  - . disjunkcia - $\vee$, |
  - . implikácia - $\rightarrow$, $\Rightarrow$, $=>$, $\supset$
  - . ekvivalencia - $\leftrightarrow$, $\Leftrightarrow$, $<=>$, $\equiv$
- predikátové symboly: veľké písmená abecedy $A - Z$
- symboly pre indivíduové premenné: malé písmená abecedy $k - z$, prípadne s indexmi
- funkčné symboly: malé písmená abecedy $f - j$
- symboly pre indivíduové konštanty: malé písmená abecedy $a - e$, prípadne s indexmi
- pomocné symboly: zátvorky - ( ) [ ]

Pravidlami gramatiky sú:

- $S \longrightarrow A$
- $A \longrightarrow QA \mid NA \mid P \mid POA \mid (A) \mid (A)OA$
- $Q \longrightarrow \forall V \mid \exists V \mid (Q)$
- $N \longrightarrow$ negácia
- $O \longrightarrow$ konjunkcia, disjunkcia, implikácia, ekvivalencia
- $P \longrightarrow X(T)$
- $X \longrightarrow$ predikátové symboly
- $T \longrightarrow F \mid V \mid C \mid F,T \mid V,T \mid C,T$
- $F \longrightarrow Y(T)$
- $Y \longrightarrow$ funkčné symboly
- $V \longrightarrow$ symboly pre indivíduové premenné
- $C \longrightarrow$ symboly pre indivíduové konštanty

V aplikácii je syntaktická analýza naimplementovaná v troch triedach, a to `CheckModel`, `InputParsing` a `CharacterCheck`. Trieda `CharcterCheck` obsahuje metódy pre zistenie, čo konkrétny znak pri parsovaní znamená. Rozoznáva či je vstup predikátovým symbolom, kvantifikátorom, logickým operátorom, funkciou, premennou alebo konštantou. Reprezentuje pravidlá gramatiky pre neterminály N, O, X, Y, V a C. Ukážka jednej z týchto metód sa nachádza vo výpise 2.

```
public bool IsVariable(char character)
{
    if (character >= 'k' && character <= 'z')
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Výpis 2: Príklad metódy z triedy `CharacterCheck`, ktorá overuje či je znak premennou

Trieda `InputParsing` slúži na rozparsovanie reťazca typu `string` zadaného používateľom do tokenov, na základe výstupov metód z triedy `CharacterCheck`. Kontroluje syntax jednotlivých tokenov, ale nezaoberá sa ich správnym umiestením v rámci formuly. Okrem správnej syntaxe taktiež kontroluje, či sa v reťazci nenachádza viacero predikátov s rovnakým menom, ale s rozdielnou aritou, pr. $P(x)$ a $P(x, y)$. V takom prípade, reťazec nie je formulou predikátovej logiky, aj keby spĺňoval všetky ostatné podmienky.

Objekty, ktoré tokeny reprezentujú, sú kvantifikátory, predikáty, operátory, zátvorky a funkcie. Informácie o nich sú uložené v instanciách triedy `ListNode`. `ListNode` reprezentuje tokeny ako uzly v obojsmernom lineárnom zozname. Každá instancia si uchováva nasledujúce informácie:

- `ListNode previous`: odkaz na predchádzajúci uzol
- `ListNode next`: odkaz na nasledujúci uzol:
- `string value`: vlatná hodnotu
- `Enums.NT type`: typ tokenu
- `int level`: level v rámci formule

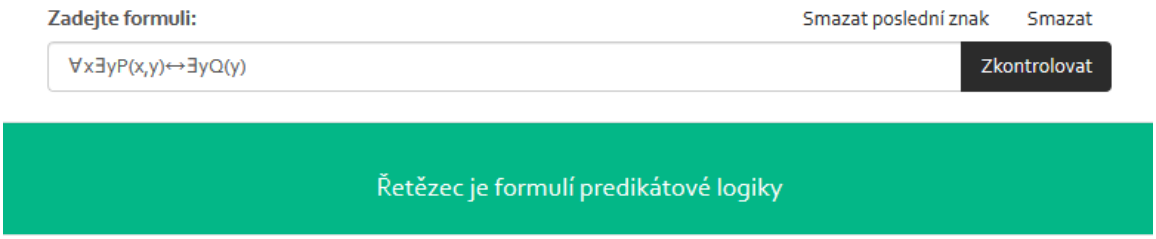
Typ objektu sa definuje kvôli jednoduchšiemu rozoznaniu o aký token ide, pri ďalšej práci s formulami. Je definovaný pomocou enumu `Enums.NT` a môže nadobúdať jednu z množiny hodnôt $\{Q,\ P,\ O,\ LP,\ RP,\ N,\ F,\ V,\ C\}$. Level tokenu sa určuje podľa zanorenia v zátvorkách

a priority. Zanorenie v zátvorkách sa ráta podľa počtu otvorených zátvoriek pred daným objektom. Pre konkrétne objekty level vyzerá nasledovne:

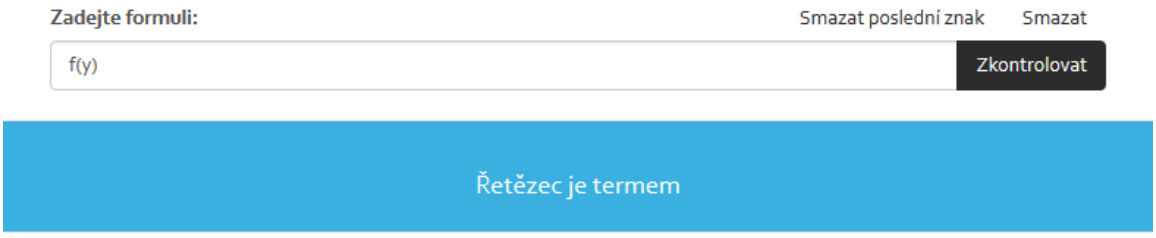
- operátory: zanorenie + 1
- kvantifikátory: zanorenie + 2
- predikáty: zanorenie + 3

Trieda `CheckModel` implementuje zvyšné pravidlá bezkontextovej gramatiky, a teda sa zaoberá hlavne správnym umiestnením objektov vo formule. Metóda `string ParseS()` predstavuje pravidlo pre počiatočný neterminál S. Metóda `void ParseA()` reprezentuje pravidlo pre neterminál A a metóda `void ParseT(string element, int i, bool insideFunction)` spája pravidlá pre neterminály P, T a F.

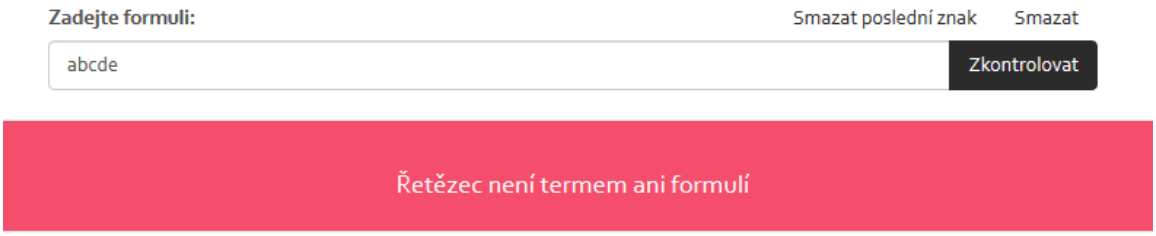
Používateľské rozhranie pre syntaktickú kontrolu umožňuje zadať reťazec ručne, s možným využitím pomocných tlačidiel, alebo vybrať niektorý z preddefinovaných príkladov. Po zadaní vstupu a kliknutí na tlačidlo *Zkontrolovat* sa zobrazí výsledok syntaktickej analýzy. Ak je reťazec formulou predikátovej logiky, tak sa zobrazí hláška *Řetězec je formulí predikátové logiky*, viď Obr.4. V prípade, že reťazec je len termom, zobrazí sa hláška *Řetězec je termem*, viď Obr.5. Ak zadaný reťazec nie je ani termom, ani formulou, zobrazí sa hláška *Řetězec není termem ani formulí*, viď Obr.6, pre niektoré chyby aj s vysvetlením prečo.



Obr. 4: Hláška, ktorá sa zobrazí v prípade, že reťazec je formulou



Obr. 5: Hláška, ktorá sa zobrazí v prípade, že reťazec je termom



Obr. 6: Hláška, ktorá sa zobrazí v prípade, že reťazec nie je formulou ani termom

8.2 Vytvorenie syntaktického stromu

Druhá časť webovej aplikácie spočíva vo vytváraní syntaktických stromov z formúl predikátovej logiky zadaných používateľom. Syntaktický strom, nazývaný aj derivačný alebo parsovací, je strom, ktorý reprezentuje syntaktickú štruktúru konkrétneho slovného reťazca podľa danej formálnej gramatiky [21]. Skladá sa z uzlov a vetví. Počiatočný uzol sa nazýva koreň a koncové uzly sú listy.

V aplikácii je vytváranie syntaktického stromu z formule predikátovej logiky implementované v triede `TreeModel` a `TreeNode`. Pomocou triedy `TreeNode` sa udržuje štruktúra uzlov stromu. Každá instancia tejto triedy obsahuje päť atribútov:

- `TreeNode parent`: odkaz na predchádzajúci uzol, rodiča
- `TreeNode child1`: odkaz na prvý nasledujúci uzol, prvého potomka
- `TreeNode child2`: odkaz na druhý nasledujúci uzol, druhého potomka
- `string value`: vlastná hodnotu
- `int level`: level v rámci formule
- `Enums.NT type`: typ uzlu

Trieda `TreeModel` sa zaoberá samotným vytváraním stromu. Predtým ako sa môže s vytváraním stromu začať, vykoná sa syntaktická kontrola, aby sa zistilo či je zadaný reťazec formulou predikátovej logiky. Keďže syntaktická kontrola je už naimplementovaná, viď podkapitola 8.1, nie je potrebné to urobiť znovu. Pomocou atribútu `private CheckModel chm` je možné z triedy `TreeModel` zavolať metódu `string CheckInputFromDiffModel(string input)`, ktorá vráti výsledok syntaktickej kontroly, viď Výpis 3.

Táto metóda má vstupný atribút `input` reprezentujúci používateľský vstup a vrácia odpoveď o tom, či je zadaný reťazec formulou predikátovej logiky alebo nie. V prípade, že je reťazec formulou, zavolá sa metóda `void CreateTree()`, ktorá začne s vytváraním stromu. V opačnom prípade sa používateľovi vypíše chybová hláška a strom sa nevytvorí.

```
public string CheckInputFromDiffModel(string input)
{
    Input = input;
    return CheckInputCharacters();
}
```

Výpis 3: Metóda `string CheckInputFromDiffModel(string input)`

Metóda `void CreateTree()` slúži na vytvorenie stromu z lineárneho zoznamu reprezentujúceho formulu, viď Výpis 4. Lineárny zoznam je postupne prechádzaný v cykle, od počiatočného uzlu po koncový. Vytváranie stromu zavísi od vzťahu medzi uzlami a od ich levelov.

Atribút `TreeNode treeNode` reprezentuje uzol v strome a atribút `ListNode listNode` reprezentuje uzol v lineárnom zozname, ktorý sa bude na uzol v strome prevádzať.

Metóda `TreeNode InsertNode(TreeNode node, string value, int level, Enums.NT type)` slúži na vkladanie uzlu do stromu. Jej atribút `TreeNode node` reprezentuje predchádzajúci vložený uzol, `string value` je hodnotou, `int level` je levelom a `Enums.NT type` je typom uzlu zo zoznamu, z ktorého sa bude vytvárať uzol stromu. Uzol stromu sa vytvorí pomocou konštruktora triedy `TreeNode` a vloží sa za posledný vložený uzol `node`. Vloženie sa vykoná pomocou priradenia správneho uzlu argumentom `parent` a `child1`, prípadne `child2`.

V prípade, že je uzol predikátom, je možné ho ďalej rozložiť na termy. Rozložiť je možné len unárne a binárne predikáty, a to kvôli zložitosti vykresľovania stromu.

V prípade, že uzol, ktorý bol vložený je operátorom, zavolá sa aj metóda `TreeNode FindNode (TreeNode node)`. Pomocou nej je možné zistiť, či je uzol zadaný ako vstupný argument potrebné presunúť v strome vyššie. Metóda vráti najvyšší uzol v strome, ktorý má väčší level ako vstupný argument. Ak žiadny taký uzol nie je, vráti vstup.

Ak je potrebné presunúť posledný vložený uzol, teda metóda `TreeNode FindNode (TreeNode node)` vrátila uzol iný od vstupného, zavolá sa metóda `TreeNode MoveNode(TreeNode node, TreeNode nodeToMoveDown)`. Táto metóda presunie uzol `TreeNode node` pred uzol `TreeNode nodeToMoveDown`. Uzol `TreeNode nodeToMoveDown` sa stane prvým potomkom, teda atribútom `child1`, uzlu `TreeNode node`.

Po vytvorení celého stromu sa zavolá metóda `void FindRoot(TreeNode node)`, pomocou ktorej sa nájde koreň stromu. Od uzlu `node`, zadaného ako vstup, sa ku koreňu dostane posunom v strome nahor (`node = node.parent`). Táto metóda slúži na to, aby sa strom začal vykresľovať od správneho uzlu.

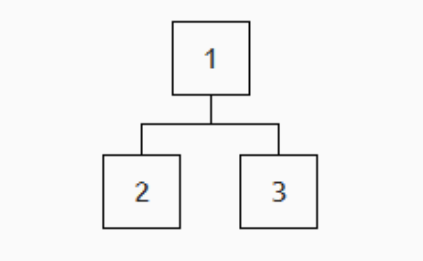
```
public void CreateTree()
{
    TreeNode treeNode = new TreeNode();
    ListNode listNode = firstNode;
    while (listNode != null)
    {
        if (!listNode.type.Equals(Enums.NT.LP) && !listNode.type.Equals(Enums.
            NT.RP))
        {
            int arity = u.CheckArity(listNode.value);
            if (listNode.type.Equals(Enums.NT.P) && arity <= 2 &&
                ParsePredicates == "yes")
            {
                treeNode = InsertNode(treeNode, listNode.value[0].ToString(),
                    listNode.level, listNode.type);
                treeNode = ParsePredicate(treeNode, listNode.value, treeNode.
                    level + 1, 2, arity);
            }
            else
            {
                treeNode = InsertNode(treeNode, listNode.value, listNode.level,
                    listNode.type);
            }
        }
        if (listNode.type.Equals(Enums.NT.O))
        {
            TreeNode nodeToMoveDown = FindNode(treeNode);

            if (!nodeToMoveDown.Equals(treeNode))
            {
                treeNode = MoveNode(treeNode, nodeToMoveDown);
            }
        }
        listNode = listNode.next;
    }
    FindRoot(treeNode);
}
```

Výpis 4: Metóda `void CreateTree()`

Syntaktický strom sa vykresľuje s využitím CSS knižnice Treeflex [22]. Knižnica sa do projektu načítava pomocou HTML príkazu <`link rel="stylesheet"href="https://unpkg.com/treeflex/dist /css/treeflexċss">` v súbore `_Layout.cshtml`. Príklad štruktúry pre tvorbu stromu sa nachádza vo výpise 5. Strom, ktorý vznikne pomocou tejto konkrétnej štruktúry je na obrázku 7.

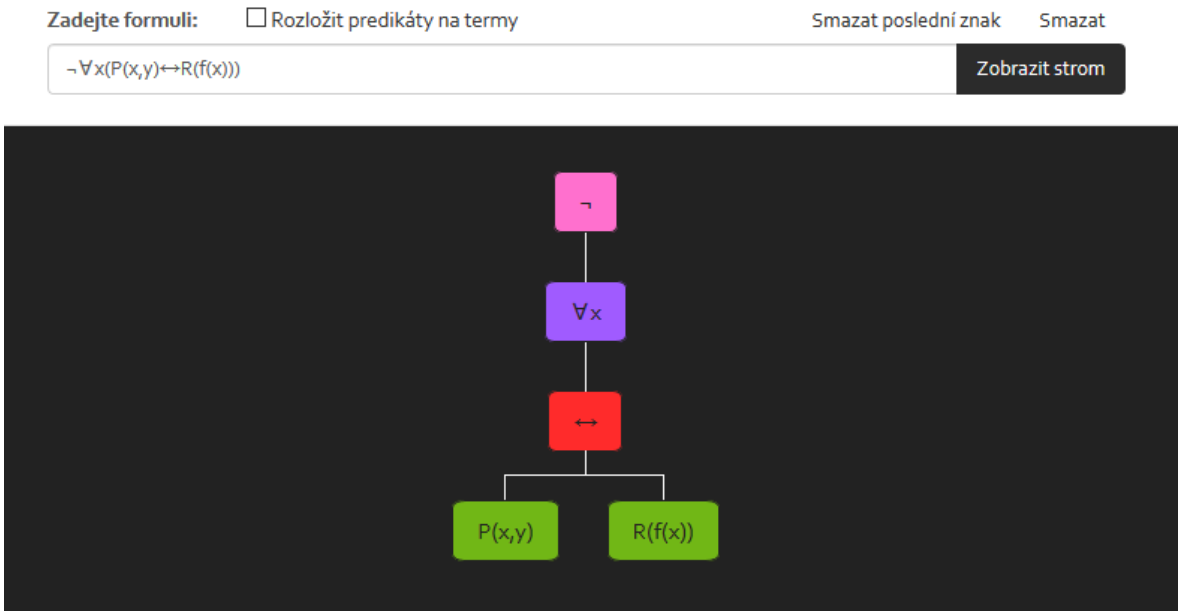
```
<div class="tf-tree">
  <ul>
    <li>
      <span class="tf-nc">1</span>
      <ul>
        <li><span class="tf-nc">2</span></li>
        <li><span class="tf-nc">3</span></li>
      </ul>
    </li>
  </ul>
</div>
```

Výpis 5: Príklad štruktúry pre vykreslenie stromu pomocou knižnice Treeflex

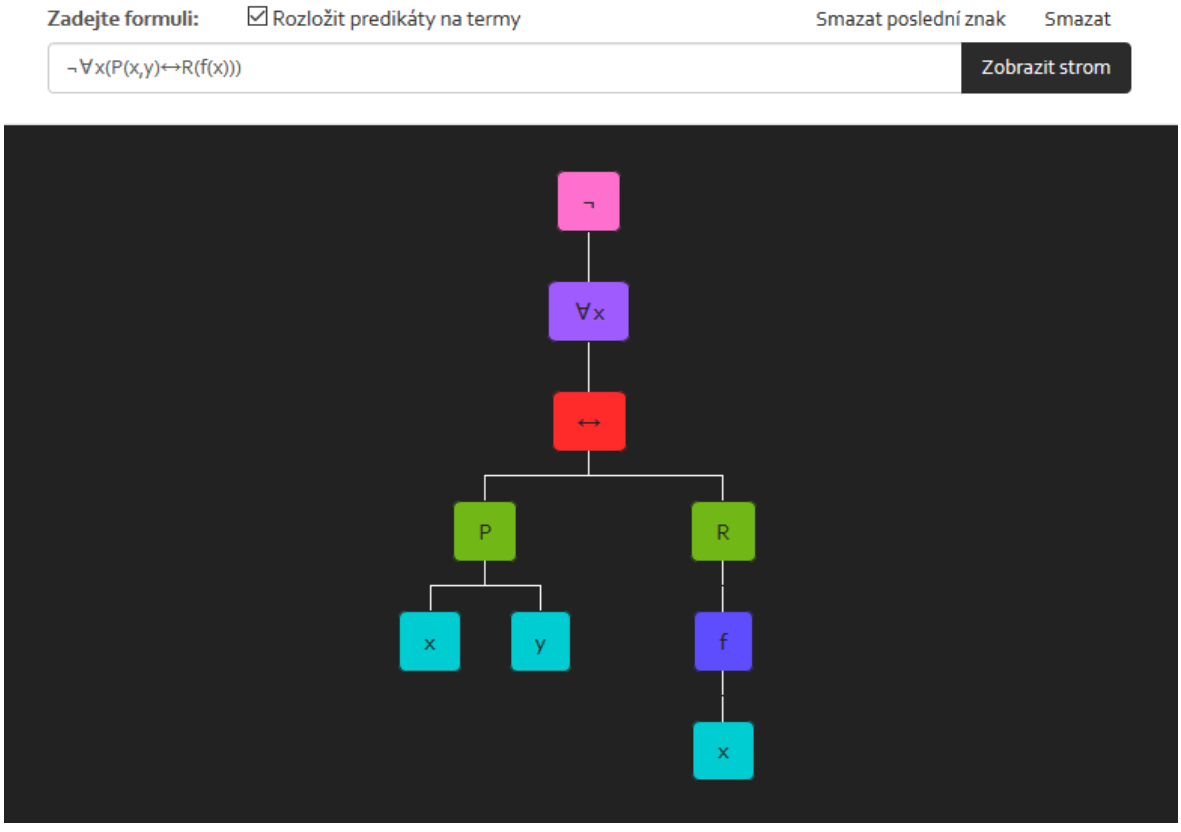


Obr. 7: Príklad zobrazeného stromu pomocou knižnice Treeflex

V používateľskom rozhraní aplikácie je možné vykresliť syntaktický strom dvomi spôsobmi. V prípade, že nie je zaškrtnuré políčko *Rozložit predikáty*, listami stromu sú celé predikáty, viď Obr.8. V prípade, že je políčko zaškrtnuté, unárne a binárne predikáty sa rozložia na termy, viď Obr.9. Funkcie sa rozložia taktiež v prípade, že ich arita je maximálne 2. Predikáty a funkcie s vyššou aritou ostanú ako celok.



Obr. 8: Príklad zobrazenia syntaktického stromu s predikátmi ako celok

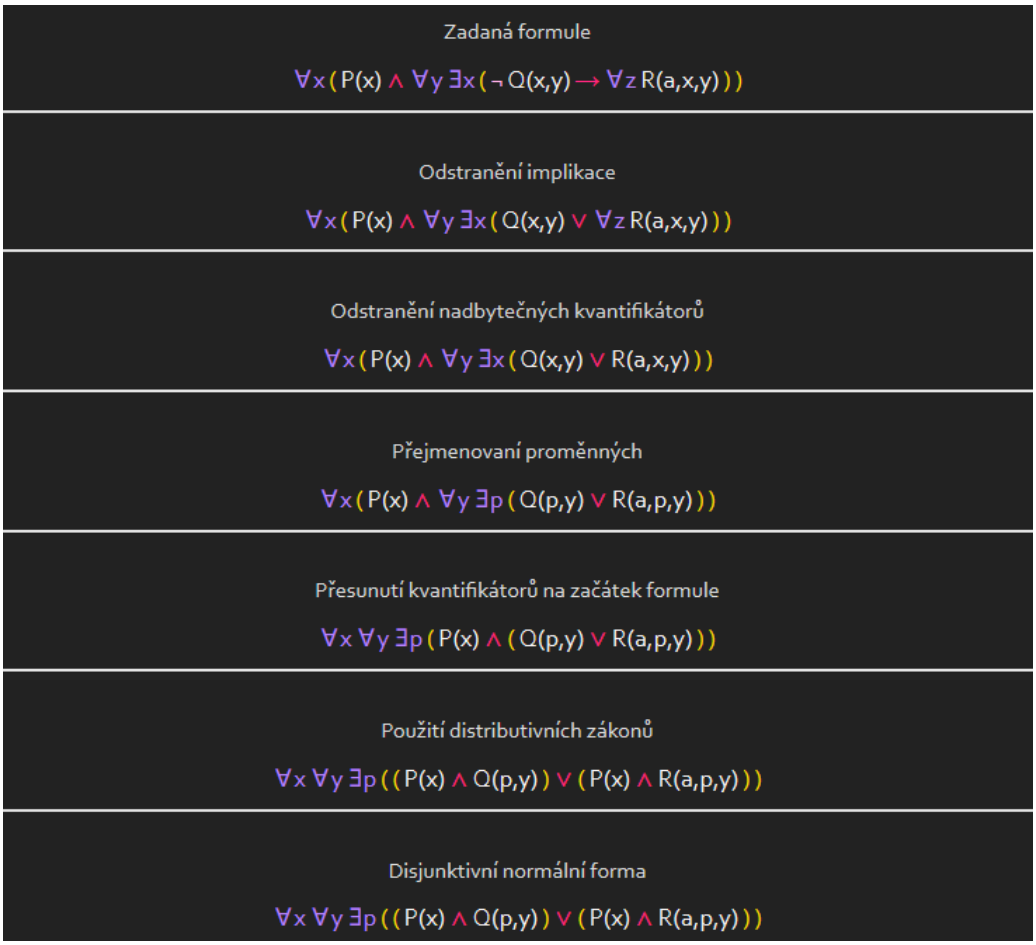


Obr. 9: Príklad zobrazenia syntaktického stromu s predikátmi rozloženými na termy

8.3 Prevod formúl do PNF

Tretia časť webovej aplikácie sa zaoberá prevodom formúl predikátovej logiky do prenexovej normálnej formy. To ako prenexová normálna forma vyzerá a všeobecný postup, akým je možné do nej formule prevádzať, je opísané v podkapitole 6.3.

Používateľské rozhranie stránky pre prevod do PNF umožňuje, rovnako ako v predchádzajúcich častiach, zadať reťazec ručne s pomocou tlačidiel alebo vybrať jeden z príkladov. Ďalej umožňuje používateľovi zvoliť či normálna forma, do ktorej má byť prevedená časť formule za kvantifikátormi, bude konjunktívna alebo disjunktívna. Po zadaní reťazca, vybraní možnosti normálnej formy a stlačení tlačidla *Převést formuli* môžu nastať dve možnosti. V prípade, že je zadaný reťazec formulou, postup prevodu sa rozpíše v krokoch podľa jednotlivých vykonaných operácií, viď Obr.10. V opačnom prípade sa vypíše chybová hláška.



Obr. 10: Príklad postupu prevodu formule do PNF

Prevod formúl do PNF je implementovaný v triede `PrenexFormModel`. Rovnako ako pri syntaktickom strome, najprv sa pomocou syntaktickej kontroly implementovanej v triede `CheckModel` overí, či je používateľom zadaný reťazec formulou predikátovej logiky. Ak je vstup formulou, za-

volá sa metóda `void CreatePrenexForm()`, ktorá volá ďalšie metódy reprezentujúce potrebné ekvivalentné úpravy. Sú nimi metódy:

- `RemoveRedundantPars()`: odstránenie nadbytočných zátvoriek, pr. $((P(x) \leftrightarrow R(x))) \Rightarrow (P(x) \leftrightarrow R(x))$, $(P(x)) \Rightarrow P(x)$
- `void RemoveEq()`: odstránenie ekvivalencií
- `void RemoveImpl()`: odstránenie implikácií
- `void DeMorganOp()`: použitie De Morganových zákonov pre operátory
- `void DeMorganQ()`: použitie De Morganových zákonov pre kvantifikátory (presunutie negácií za kvantifikátory)
- `void RenameVariables()`: premenovanie voľných a viazaných premenných
- `void MoveQuantifiers()`: použitie zákonov pre prevod do prenexového tvaru (presuntie kvantifikátorov na začiatok formule)
- `void CreateCNFOrDNF()`: prevedenie časti formule za kvantifikátormi do KNF alebo DNF, podľa používateľského vstupu, s využitím distributívnych a asociatívnych zákonov

Metódy pracujú s formulou v forme obojsmerného lineárneho zoznamu, ktorý vzniká pri syntaktickej kontrole.

8.3.1 Odstránenie ekvivalencií

Pre prevod formúl predikátovej logiky do ekvivalentného tvaru bez ekvivalencií sú implementované dva zákony z podkapitoly 5.6, konkrétne $\neg(X \leftrightarrow Y) \Leftrightarrow (X \wedge \neg Y) \vee (\neg X \wedge Y)$ a $(X \leftrightarrow Y) \Leftrightarrow (X \rightarrow Y) \wedge (Y \rightarrow X)$. Implementácia začína v metóde `RemoveEq()` a je možné ju zhrnúť do nasledujúcich bodov:

1. nájdenie prvého uzlu `node` reprezentujúceho ekvivalenciu (v prípade, že tam žiadny nie je, nasledujúci postup sa nevykoná)
2. identifikovanie argumentov ekvivalencie X a Y
   (a) argument X sa získava rekurzívne prechádzaním uzlov od ekvivalencie, smerom k začiatku zoznamu (`temp = temp.previous`)
   (b) cyklus sa zastaví, keď predchádzajúci uzol aktuálneho uzlu `temp` je `null` alebo ľavá zátvorka s levelom o 1 nižším než má ekvivalencia
   (c) argument X tvoria všetky uzly od uzlu `temp` až po uzol predchádzajúci ekvivalencii, vrátane ich oboch (argument X môže byť tvorený aj len jedným uzlom)

(d) argument Y sa získava rekurzívne prechádzaním uzlov od ekvivalencie, smerom ku koncu zoznamu (`temp = temp.next`)

(e) cyklus sa zastaví, keď nasledujúci uzol aktuálneho uzlu `temp` je `null` alebo pravá zátvorka s levelom o 2 nižším než má ekvivalencia

(f) argument Y tvoria všetky uzly od uzlu nasledujúceho ekvivalencii až po uzol `temp`, vrátane ich oboch (argument Y môže byť tvorený aj len jedným uzlom)

3. vytvorenie kópií obidvoch členov X aj Y

4. v prípade, že sa pred členom `temp` nachádza ľavá zátvorka s levelom o 1 nižším než má ekvivalencia a pred ňou je negácia, teda jedná sa o prvý zákon (negácia ekvivalencie), vykonajú sa kroky

   (a) odstránenie negácie pred zátvorkou, v ktorej sa nachádza uzol `node`

   (b) zmena pôvodnej hodnoty uzlu `node` z ekvivalencie na konjunkciu

   (c) zmena negácie pôvodného druhého argumentu Y pomocou metódy `void Change Negation(ListNode node)` (odstránenie negácie ak je pred argumentom, pridanie ak nie je)

   (d) ozátvorkovanie prvého nového člena (pôvodné X a Y spojené uzlom `node`), vznikne člen A $(X \wedge \neg Y)$

   (e) zmena negácie prvého skopírovaného argumentu X

   (f) vytvorenie nového uzlu reprezentujúceho konjukciu a jeho pridanie medzi skopírované argumenty X a Y

   (g) ozátvorkovanie druhého nového člena (skopírované X a Y spojené novovytvorenou konjunkciou), vznikne člen B $(\neg X \wedge Y)$

   (h) vytvorenie nového uzlu reprezentujúceho disjunkciu a jeho pridanie medzi nové členy A a B, vznikne výsledný tvar $(X \wedge \neg Y) \vee (\neg X \wedge Y)$

5. v ostatných prípadoch, teda keď sa jedná o druhý zákon, vykonajú sa kroky

   (a) zmena pôvodnej hodnoty uzlu `node` z ekvivalencie na implikáciu

   (b) ozátvorkovanie prvého nového člena (pôvodné X a Y spojené uzlom `node`), vznikne člen A $(X \rightarrow Y)$

   (c) vytvorenie nového uzlu reprezentujúceho implikáciu a jeho pridanie medzi skopírované argumenty X a Y v opačnom poradí

   (d) ozátvorkovanie druhého nového člena (skopírované X a Y spojené novovytvorenou implikáciou), vznikne člen B $(Y \rightarrow X)$

   (e) vytvorenie nového uzlu reprezentujúceho konjunkciu a jeho pridanie medzi nové členy A a B, vznikne výsledný tvar $(X \rightarrow Y) \wedge (Y \rightarrow X)$

6. opakovanie postupu v prípade, že formula ešte obsahuje nejakú ďalšiu ekvivalenciu

8.3.2 Odstránenie implikácií

Prevod formúl do ekvivalentného tvaru bez implikácií je implementovaný v metóde `RemoveImpl()`. Základom sú zákony pre odstránenie implikácii z podkapitoly 5.6. Implementáciu postupu je možné popísať v nasledujúcich bodoch:

1. nájdenie prvého uzlu `node` reprezentujúceho implikáciu (v prípade, že tam žiadny nie je, nasledujúci postup sa nevykoná)
2. identifikovanie argumentov implikácie, X a Y, funguje rovnako ako pri odstraňovaní ekvivalencií v podkapitole 8.3.1
3. v prípade, že sa jedná o prvý zákon (negácia implikácie)
   (a) odstránenie negácie pred zátvorkou, v ktorej sa nachádza uzol `node`
   (b) zmena pôvodnej hodnoty uzlu `node` z implikácie na konjunkciu
   (c) zmena negácie druhého argumentu Y, vznikne $(X \wedge \neg Y)$
4. v prípade, že sa jedná o druhý zákon
   (a) zmena pôvodnej hodnoty uzlu `node` z implikácie na konjunkciu na disjunkciu
   (b) zmena negácie prvého argumentu X, vznikne $(\neg X \vee Y)$
5. opakovanie postupu v prípade, že formula ešte obsahuje nejakú ďalšiu implikáciu

8.3.3 Použitie De Morganových zákonov

Na presunutie negácie do vnútra zátvorky sa využívajú De Morganove zákony pre operátory, definované v podkapitole 5.5. Implementácia týchto zákonov v metóde DeMorganOp() vyzerá nasledovne:

1. nájdenie uzlu, ktorý má ako hodnotu negáciu, a jeho nasledujúci uzol je ľavá zátvorka
2. odstránenie negácie pred zátvorkou
3. nájdenie posledného uzlu `node`, ktorým reprezentuje konjunkciu alebo disjunkciu, s levelom o 1 vyšším ako má ľavá zátvorka
4. identifikovanie argumentov konjunkcie alebo disjunkcie, X a Y, funguje rovnako ako pri odstraňovaní ekvivalencií v podkapitole 8.3.1
5. zmena negácie prvého argumentu X
6. zmena negácie druhého argumentu Y
7. zmena hodnoty uzlu `node` pomocou metódy `string InvertOperator(string ope)` (konjunkciu na disjunkciu, a naopak)

8. opakovanie postupu v prípade, že sa vo formule nachádza negácia pred nejakou ďalšou zátvorkou

Presunutie negácie za kvantifikátory sa vykonáva taktiež pomocou De Morganových zákonov, ale pre kvantifikátory. Ich implementáciu v metóde `void DeMorganQ()` je možné popísať v bodoch:

1. nájdenie uzlu reprezentujúceho kvantifikátor, ktorého predchádzajúcim uzlom je negácia
2. presunutie negácie spred kvanantifikátora za neho
3. inverzia kvantifikátora pomocou metódy `string InvertQuantifier(string quantifier)` (existenčný na všeobecný, a naopak)
4. opakovanie v prípade, že sa negácia nachádza bezprostredne pre nejakým ďalším kvantifikátorm

8.3.4 Premenovanie premenných a odstránenie nadbytočných kvantifikátorov

Ďalším krokom pri prevode formúl do PNF je premenovanie premenných a odstránenie nadbytočných kvantifikátorov. Premenné sa vo formule musia premenovať tak, aby žiadna voľná alebo viazaná premenná nemala rovnaké meno. Keby sa premenné nepremenovali, zákony prevodu do prenexového tvaru by nemuselo byť možné použiť. Keby sa použili nesprávne formula by po úpravách mala iný význam.

Implementácia týchto ekvivalentných úprav začína v metóde `void RenameVariables()`. Postup premenovania premenných a odstránenia kvantifikátorov je nasledovný:

1. vytvorenie kolekcie `Dictionary<ListNode, List<ListNode» quantsWithPreds`, v ktorej sú každému kvantifikátoru priradené predikáty, na ktoré sa vzťahuje
2. vytvorenie kolekcie `Dictionary<ListNode, List<ListNode» predsWithQuants`, v ktorej sú každému predikátu priradené kvantifikátory, ktoré sa na neho vzťahujú
3. vytvorenie zoznamu `List<string> checkedVariables`, ktorý bude obsahovať premenné, ktoré už nie je možné použiť inde vo formule
4. na základe hodnôt v prvej kolekcii, je možné identifikovať kvantifikátory, ktoré sú nadbytočné, teda sa nevzťahujú na žiadny predikát (o niektorých kvantifikátoroch sa zistí, že sú nadbytočné až po premenovaní, a teda sa odstránia až vtedy)
5. nadbytočné kvantifikátory sú z formule odstránené pomocou metódy `void RemoveUnused Quants(Dictionary<ListNode, List<ListNode» quantsWithPreds)`

6. po odstránení kvantifikátorov sa zavolá metóda `void RenameFreeVars(Dictionary<List Node, List<ListNode» predsWithQuants)`, ktorá je na základe hodnôt v druhej kolekcii, schopná identifikovať predikáty obsahujúce len voľné premenné (v kolekcii sa nenachádzajú)

7. rekurzívne sa prejdú všetky nájdené predikáty a môže nastať jedna z dvoch možností

   (a) premenná sa v zozname `checkedVariables` nenachádza, teda sa tam pridá, a pokračuje sa k ďalšej voľnej premennej

   (b) premenná sa v zozname `checkedVariables` nachádza, preto je potrebné ju najprv premenovať pomocou metódy `string ChangeVariable(char letter, string varNum, List<string> nextQuantVars)`, po premenovaní sa zmenená hodnota pridá do zoznamu `checkedVariables` a pokračuje sa k ďalšej premennej

8. ako ďalšia sa zavolá metóda `void RenameBoundVars(Dictionary<ListNode, List<ListNode» quantsWithPreds, Dictionary<ListNode, List<ListNode» predsWithQuants)`, ktorá sa stará o premenovanie voľných a viazaných premenných v zvyšných predikátoch a kvantifikátoroch

9. rekurzívne sa prejdú všetky kvantifikátory a môže nastať jedna z dvoch možností pri premenovávaní viazaných premenných

   (a) premenná pri kvantifikátore sa v zozname `checkedVariables` nenachádza, teda sa tam pridá, a pokračuje sa ďalej

   (b) premenná pri kvantifikátore sa v zozname `checkedVariables` nachádza, preto je potrebné ju najprv premenovať pred pridaním do zoznamu, premenná sa premenuje aj vo všetkých predikátoch, na ktoré sa kvantifikátor priamo vzťahuje

10. s premenovaním viazaných premenných sa v predikátoch zároveň premenúvajú aj zvyšné voľné premenné rovnakým spôsobom

8.3.5 Presunutie kvantifikátorov na začiatok formule

Po premenovaní premenných si môžeme byť istý, že sa vo formule nenachádza žiadna voľná a viazaná s rovnakým menom. Preto použitie zákonov, z podkapitoly 5.7, pre prevod formule do prenexového tvaru nie je problematické. Implementácia týchto zákonov sa nachádza `void MoveQuantifiers()` a vyzerá nasledovne:

1. nájdenie od začiatku najbližšieho doteraz nepresunutého kvantifikátora

   (a) v prípade, že je prvý, presunie sa na úplný začiatok formule

   (b) v ostatných prípadoch, presunie sa na začiatok za predchádzajúci presunutý kvantifikátor

2. opakovanie dokým nie sú všetky kvantifikátory presunuté na začiatok formule

8.3.6 Použitie distrubutívnych a asociatívnych zákonov

Aby sa z formuly v prenexovom tvare stala formula v prenexovej normálnej forme, je potrebné aby časť formule za kvantifikátormi bola v konjunktívnej alebo disjunktívnej normálnej forme. K vytvoreniu KNF alebo DNF už smerovali predchádzajúce ekvivalentné úpravy. V niektorých prípadoch je možné, aby formula už v tejto chvíli bola v PNF, v iných je potrebné ešte použiť distributívne a asociatívne zákony definované v podkapitolách 5.4 a 5.3.

Posledná metóda `void CreateCNFOrDNF()` volaná z `void CreatePrenexForm()` určí či bude výsledná formula v KNF alebo DNF, a to podľa možnosti, ktorú zvolil používateľ. V prípade KNF zavolá metódu `void CreateCNF(ListNode mainOpe)` a v prípade DNF `void CreateDNF(List Node mainOpe)`. Vstupný atribút `ListNode mainOpe` je posledný uzol v lineárnom zozname reprezentujúcom formulu, ktorého hodnota je operátor a jeho level je najmenší medzi všetkými operátormi. Na jeho nájdenie slúži metóda `ListNode FindMainOpe()`.

Distributívne zákony sú implementované v metódach `void DistributiveLawABAC(ListNode mainOpe, string firstOpe, string secondOpe)` a `void DistributiveLawACBC( ListNode mainOpe, string firstOpe, string secondOpe)`. Obidve metódy fungujú na rovnakom princípe, ale majú iné vstupné argumenty. Prvá metóda implementuje zákony v tvare:

- $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
- $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Druhá metóda implementuje zákony v tvare definovanom v spomínanej podkapitole 5.4, teda:

- $(A \wedge B) \vee C \Leftrightarrow (A \vee C) \wedge (B \vee C)$
- $(A \vee B) \wedge C \Leftrightarrow (A \wedge C) \vee (B \wedge C)$

Použitím asociatívnych zákonov sa z formule odstránia nepotrebné zátvorky, v závislosti na zvolenej normálnej forme. Sú implementované v metóde `void AssociativeLaw()`. Po tomto kroku sa formula už určite nachádza v prenexovej normálnej forme.

8.4 Pravdivosť formúl

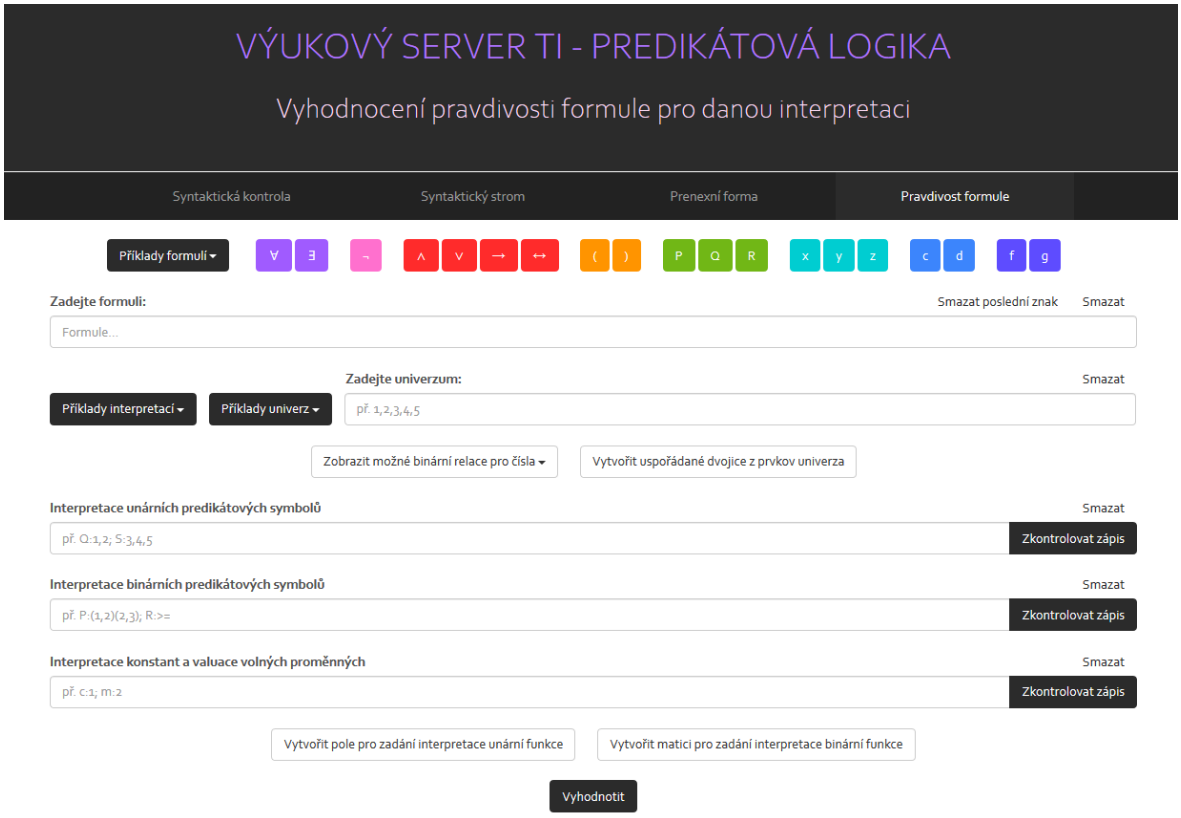
Posledná časť webovej stránky sa zaoberá vyhodnocovaním pravdivosti formúl v závislosti na zadaných interpretáciách. Základný postup interpretovania formúl spolu s definíciami potrebných pojmov sa nachádza v podkapitole 4.2.5.

Používateľské rozhranie pre vyhodnocovanie pravdivosti formúl tvorí niekoľko vstupných polí a tlačidiel, viď Obr.11. Sú nimi:

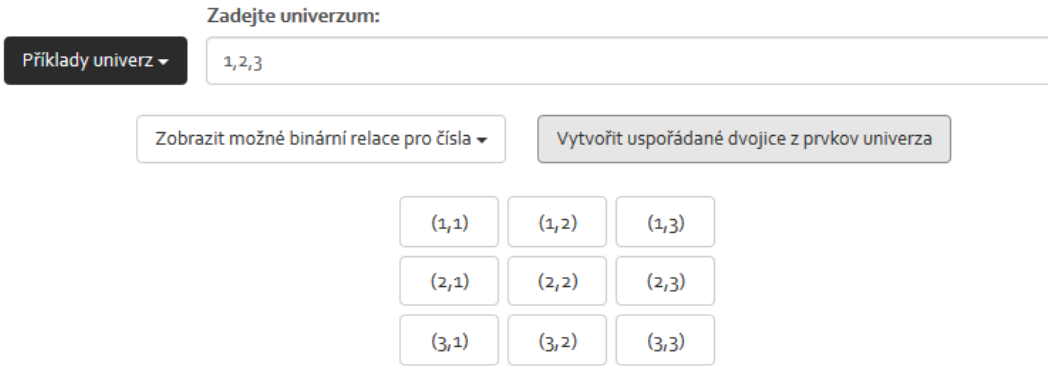
- pole pre zadanie formule, fungujúce rovnako ako pri ostatných stránkach
- pole pre zadanie univerza s možnosťou výberu niektorého z príkladov, príklad zápisu: $1, 2, 3, 4, 5$
- pole pre zadanie interpretácií unárnych predikátových symbolov, príklad zápisu: $Q : 3, 4, 5$; $S : 1, 2$
- pole pre zadanie interpretácií binárnych predikátových symbolov, príklad zápisu: $P :>=$; $R : (1, 1)(2, 2)(3, 3))$
- pole pre zadanie interpretacií konštánt a valuácií voľných premenných, príklad zápisu: $m : 2$; $c : 4$
- tlačidlo na zobrazenie príkladov binárnych relácii ($>$, $>=$, $=$, $<=$, $<$) a vybranie niektorej ako interpretáciu binárneho predikátu, (predikát môže predstavovať niektorú z týchto relácií len v prípade, že univerzum je tvorené iba číslami, v opačnom prípade sa zobrazí chybová hláška)
- tlačidlo na vytvorenie všetkých možných usporiadaných dvojíc z prvkov univerza, viď Obr.12, ktoré je možné použiť ako prvky relácií priradených binárnym predikátom
- tlačidlo na vytvorenie polí pre zadanie interpretácií unárnych funkčných symbolov, viď Obr.13
- tlačidlo na vytvorenie matíc pre zadanie interpretácií binárnych funkčných symbolov, viď Obr.13

Interpretácie unárnych aj binárnych funkčných symbolov je možné zadať ručne alebo nechať si ich vygenerovať náhodné hodnoty z univerza kliknutím na tlačidlo *Náhodne hodnoty*. Tlačidlom *Přidat další funkci* je možné pridať polia pre zadanie interpretácie ďalšej funkcie,tlačidlom *Smazat poslední funkci* sa zmažú všetky posledné vytvorené polia aj s hodnotami a tlačidlom *Smazat vše* sa zmažú všetky interpretácie pre unárne alebo binárne funkcie.

Zadávanie a vyhodnocovanie interpretácií pre predikáty a funkcie s aritou väčšou ako 2 nie je podporované. V prípade predikátov je to z toho dôvodu, že interpretácie sa primárne zadávajú podľa usporiadaných n-tíc. Kvôli jednoduchosti zadávania hodnôt a prehľadnosti používateľského rozhrania, vytváranie viac ako dvojprvkových n-tíc z univerza by nebolo vhodné. Dôvody v prípade funkcií sú veľmi podobné. Vytváranie viac ako dvoj rozmerných matíc by bolo neprehľadné a používateľ by mohol mať problémy so správnym zadávaním hodnôt do polí.

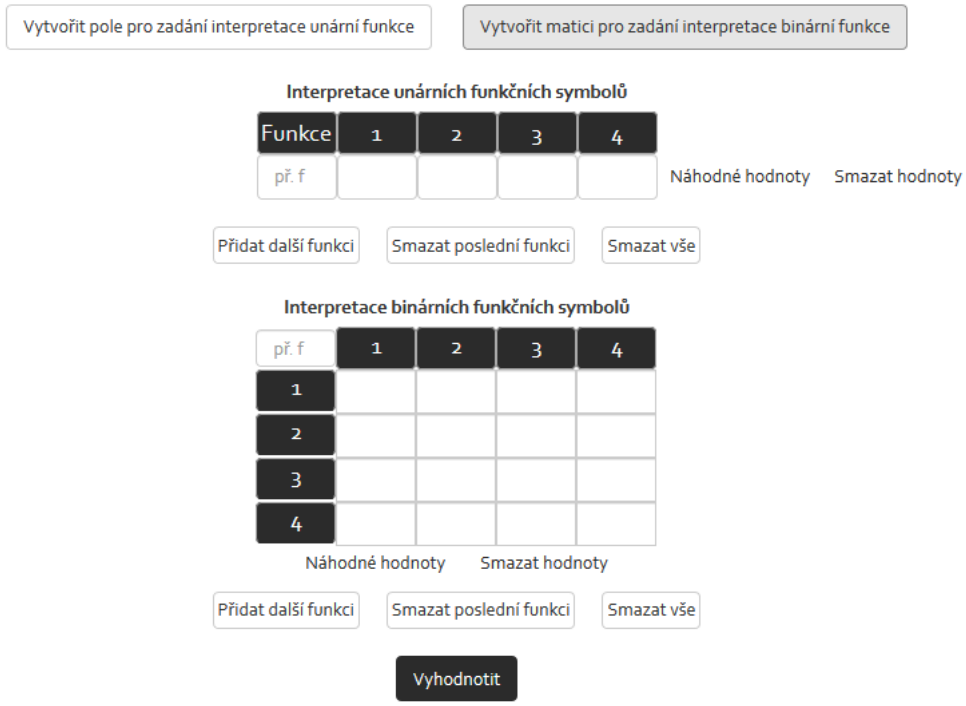


Obr. 11: Používateľské rozhranie pre vyhodnocovanie pravdivosti formúl



Obr. 12: Príklad vytvorených usporiadaných dvojíc z hodnôt univerza

Vyhodnocovanie pravdivosti formúl na základe zadaných interpretácií je naimplementované v triede `TruthValuesModel`. Najprv, rovnako ako pri predchádzajúcich častiach aplikácie, sa pomocou syntaktickej kontroly implementovanej v triede `CheckModel` overí, či je používateľom zadaný reťazec formulou predikátovej logiky. Potom sa pomocou atribútu `PrenexFormModel pfm` zavolajú metódy pre presunutie negácií za kvantifikátory, premenovanie premenných a odstránenie nadbytočných kvantifikátorov.



Obr. 13: Zobrazené polia pre zadanie interpretácií funkčných symbolov

Ďalší krok spočíva v rozparsovaní všetkých používateľom zadaných vstupov. Na parsovanie slúži metóda `string ParseInputs()`, ktorá ďalej volá metódy pre parsovanie vstupov jednotlivých polí zvlášť. V prípade, že sú všetky zadané vstupy syntakticky správne, je možné začať s vyhodnocovaním.

Implementácia začína v metóde `string EvaluateValues(Dictionary<string, int> values)`, ktorá rekurzívne mení hodnoty kvantifikovaných premenných a potom na ne volá metódu `string Evaluate(Dictionary<string, int> values)`. Atribút `Dictionary<string, int> values` obsahuje index jednej hodnoty z univerza pre každú kvantifikovanú premennú. Na začiatku je všetkým premenným priradená hodnota 0. Metóda `string Evaluate(Dictionary <string, int> values)` vyhodnocuje pravdivosť formule pre konkrétne hodnoty z univerza s využitím zadaných interpretácií a valuácií. Postup vyhodnocovania je nasledovný:

1. v každom predikáte sa zamenia premenné za hodnoty z univerza podľa indexu v kolekcii `values`
2. na základe zadaných interpretácií sa vyhodnotí pravdivostná hodnota všetkých predikátov
   (a) v metóde `string EvaluateUnPredicate(string predicate, Dictionary<string, int> values)` sa vyhodnotí pravdivostná hodnota unárnych predikátov
   (b) v metóde `string EvaluateBinPredicate(string predicate, Dictionary<string, int> values)` sa vyhodnotí pravdivostná hodnota binárnych predikátov

3. v prípade, že predikát obsahuje funkciu, zavolá sa metóda pre vyhodnotenie funkcie `string EvaluateFunction(string function, Dictionary<string, int> values)`, ktorá vráti výsledok aplikovania funkcie na premennú, v prípade zanorenia viacerých funkcií do seba sa metóda volá rekurzívne

4. po získaní pravdivostných hodnôt všetkých predikátov sa zavolá metóda `string Evaluate Operators(List<string> formula)`, ktorá na pravdivostné hodnoty aplikuje operácie, v prípade, že vo formule nejaké sú, a vráti výslednú pravdivostnú hodnotu pre konkrétne valuácie premenných případe, že je to potrebné postup sa opakuje s novými hodnotami priradenými viazaným premenným

Priraďovanie hodnôt premenným, pre ktoré sa v konkrétnom kroku bude formula vyhodnocovať, a výsledné určenie pravdivosti formule pre danú interpretáciu vyzerá nasledovne:

1. v prípade, že metóda `string EvaluateValues(Dictionary<string, int> values)` vrátila hodnotu pravda (True), nájde sa od konca najbližší všeobecný kvantifikátor a zvýši sa index hodnoty pre kvantifikovanú premennú o 1

   (a) v prípade, že obsahuje index najvyššej hodnoty univerza, hodnota sa zmení na 0 a hľadá sa ďalší najbližší všeobecný kvantifikátor

   (b) v prípade, že už žiadny taký ďalší kvantifikátor nie je, vyhodnocovanie skončí, pravdivostná hodnota formule pre danú interpretáciu je pravda a v používateľskom rozhraní sa zobrazí hláška *Formule je pro zadanou interpretaci pravdivá*, viď Obr.14.

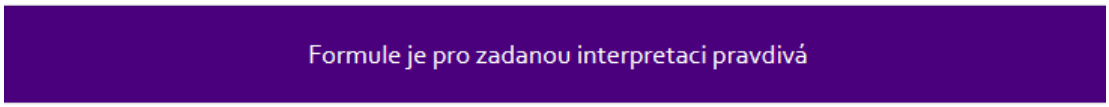
2. všetkým existenčných kvantifikátoroch, ktoré sa nachádzajú medzi spomínaným všeobecným a koncom formule, sa zmení hodnota indexu pre premennú na 0, pre ostatné sa hodnoty nezmenia

3. v prípade, že metóda `string EvaluateValues(Dictionary<string, int> values)` vrátila hodnotu nepravda (False), nájde sa od konca najbližší existenčný kvantifikátor a zvýši sa index hodnoty pre kvantifikovanú premennú o 1

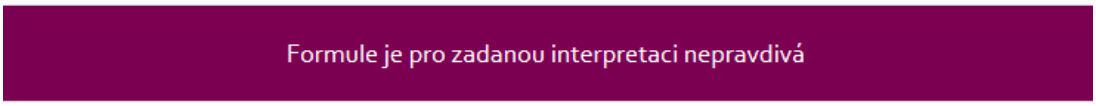
   (a) v prípade, že obsahuje index najvyššej hodnoty univerza, hodnota sa zmení na 0 a hľadá sa ďalší najbližší existenčný kvantifikátor

   (b) v prípade, že už žiadny taký ďalší kvantifikátor nie je, vyhodnocovanie skončí, pravdivostná hodnota formule pre danú interpretáciu je nepravda a v používateľskom rozhraní sa zobrazí hláška *Formule je pro zadanou interpretaci pravdivá*, viď Obr.15.

4. všetkým všeobecným kvantifikátoroch, ktoré sa nachádzajú medzi spomínaným existenčným a koncom formule, sa zmení hodnota indexu pre premennú na 0, pre ostatné sa hodnoty nezmenia



Obr. 14: Zobrazená hláška v prípade, že formula je v zadanej interpretácii pravdivá



Obr. 15: Zobrazená hláška v prípade, že formula je v zadanej interpretácii nepravdivá

8.5 Konfiguračné súbory

Predvyplnené príklady reťazcov, formúl, univerz a interpretácií sú zadané cez XML konfiguračné súbory. Tieto súbory sa nachádzajú v zložke */wwwroot/xml* a sú nimi:

- syntaxCheck.xml: príklady reťazcov pre syntaktickú kontrolu
- syntaxTree.xml: príklady formúl pre tvorbu syntaktických stromom
- prenexForm.xml: príklady formúl pre prevod do prenexovej formy
- truthValsFormulas.xml: príklady formúl pre vyhodnocovanie pravidvosti
- truthValsUniverses.xml: príklady univerz pre vyhodnocovanie pravidvosti
- truthValsInterpretss.xml: príklady interpretácií pre vyhodnocovanie pravidvosti

  V rovnakej zložke sa taktiež nachádza aj súbor *README.txt*, v ktorom je popísané ako správne do súborov zadávať hodnoty a aké špeciálne znaky je možné používať.

8.6 Spustenie projektu a nasadenie na server

Projekt pre Visual Studio, ktorý obsahuje zdrojový kód vyvinutej webovej aplikácie, sa nachádza v prílohe v archíve *PL.zip*. Kedže bol pre vývoj aplikácie použitý framework Razor Pages, na jej spustenie je potrebné mať:

- Visual Studio 2017 verziu 15.9 alebo vyššiu
- .NET Core SDK 2.2 alebo novšie

Zložka určená pre nasadenie aplikácie na server sa náchadza v archíve *Pl-publish.zip*. Pri nasadzovaní ju ako celok stačí vložiť do koreňového adresára. Webová aplikácia je aktuálne nasadená v cloudovej službe Microsoft Azure. Odkaz na ňu sa nachádza v súbore *Pl-azure.txt*.

9 Záver

Cieľom tejto diplomovej práce bolo vytvoriť webovú aplikáciu, ktorá bude umožňovať riešiť rôzne typy úloh zameraných na prácu s formulami predikátovej logiky. Konkrétne v nej malo byť možné syntakticky kontrolovať zadané reťazce, zobraziť strom na základe zadanej formule, previesť formulu do prenexového normálneho tvaru a vyhodnotiť pravdivosť formule na základe zadaných interpretácií.

Webová aplikácia poskytujúca tieto funkcionality bola vyvinutá. Jej vývoj prebiehal v prostredí Microsoft Visual Studio 2017 pomocou technológie Razor Pages. Logická vrstva aplikácie bola písaná v jazyku C# a prezentačná vrstva v jazyku Razor, ktorého základom je jazyk HTML. Vývoj aplikácie prebiehal od základov, nebola použitá žiadna už existujúca aplikácia.

Používateľské rozhranie aplikácie je rozdelené na štyri časti, tak že každej funkcionalite je venovaná jedna stránka. Vo všeobecnosti rozhranie umožňuje používateľom zadávať vlastné vstupy, na základe ktorých sa im zobrazí výsledok operácie. Počet a obsah vstupných hodnôt sa pre jednotlivé stránky líšia.

V časti syntaktickej kontroly, je používateľ po zadaní symbolov do vstupného poľa schopný zistiť, či je daný reťazec formulou predikátovej logiky, termom, alebo ani jedným z nich. Okrem vlastného zadávania reťazca si môže vybrať z preddefinovaných príkladov, v ktorých sa nájdu zástupcovia všetkých troch skupín.

Pre vykreslenie syntaktického stromu používateľ potrebuje zadať reťazec, ktorý je formulou predikátovej logiky. V opačnom prípade sa strom nevytvorí a vypíše sa len chybová hláška.

Pri prevode formule do prenexovej normálnej formy je používateľovi umožnené, okrem zadania vstupného reťazca, vybrať aj typ normálnej formy, do ktorej bude prevedená časť formule za prenexom. Má na výber z konjunktívnej a disjunktívnej normálnej formy. Rovnako ako pri vytváraní syntaktického stromu, zadaný reťazec musí byť formulou predikátovej logiky, aby sa zobrazil postup prevodu formule do PNF a nie chybová hláška. V prípade, že sa postup zobrazí, je rozdelený do krokov podľa jednotlivých vykonaných ekvivalentných úprav.

Vyhodnocovanie pravdivosti formule ponúka používateľovi možnosť zadať vstupnú formulu a k nej vlastnú interpretáciu. Formulu aj interpretáciu je taktiež možné vybrať z existujúcich príkladov. Interpretácia musí byť zadefinovaná pre všetky prvky vo formule, ktoré ju vyžadujú, inak nie je možné vyhodnotiť jej pravdivostnú hodnotu. Zadanie interpretácie je možné pre unárne a binárne predikáty, unárne a binárne funkcie a konštanty. Okrem toho, je taktiež možné zadať aj valuáciu voľných premenných.

Vytvorená webová aplikácia by sa mala stať komponentou školského výukového serveru pre teoretickú informatiku. Výukový server by mal študentom pomôcť pri riešení a pochopení úloh v predmetoch týkajúcich sa teoretickej informatiky. Vzniká v rámci diplomových a bakalárskych prác na VŠB-TUO. Aplikácia vytvorená v rámci tejto diplomovej práce sa zameriava na úlohy týkajúce sa predikátovej logiky.

V budúcnosti je možné webovú aplikáciu rozšíriť o ďalšie funkcionality, a tým zväčšiť jej pôsobnosť. Keďže predikátová logika je široký pojem, existuje ešte veľké množstvo vecí, ktoré by bolo možné naimplementovať a pridať do aplikácie.

Literatúra

[1] Encyclopædia Britannica. *Logic* [online]. Encyclopædia Britannica, inc.. Poslední změna 25.4.2018. [cit. 25.4.2019]. Dostupné z `https://www.britannica.com/topic/logic`

[2] SCHWICHTENBERG, Helmut. *Mathematical Logic* [online]. Mathematisches Institut der Universität München. 2003/2004 [cit. 25.4.2019]. Dostupné z `http://www.mathematik.uni-muenchen.de/~schwicht/lectures/logic/ws03/ml.pdf`

[3] KLEMENT, Kevin C. *Propositional Logic* [online]. The Internet Encyclopedia of Philosophy. [cit. 25.4.2019]. ISSN 2161-0002 Dostupné z `https://www.iep.utm.edu/prop-log/`

[4] Stanford University. *Chapter 2 - Propositional Logic* [online]. Stanford University. [cit. 25.4.2019]. Dostupné z `http://intrologic.stanford.edu/notes/chapter_02.html`

[5] WORREL, James. *First-Order Logic* [online]. University of Oxford, 2016. [cit. 25.4.2019]. Dostupné z `http://www.cs.ox.ac.uk/james.worrell/lecture9-2015.pdf`

[6] Stanford University. *14 Predicate Logic* [online]. Stanford University. [cit. 25.4.2019]. Dostupné z `http://infolab.stanford.edu/~ullman/focs/ch14.pdf`

[7] University of Malta. *Some Equivalence Laws of Propositional Logic* [online]. University of Malta. [cit. 25.4.2019]. Dostupné z `http://www.cs.um.edu.mt/gordon.pace/Teaching/DiscreteMaths/Laws.pdf`

[8] Encyclopedia of Mathematics. *Commutativity* [online]. Springer Verlag GmbH, European Mathematical Society. [cit. 25.4.2019]. Dostupné z `https://www.encyclopediaofmath.org/index.php/Commutativity`

[9] Encyclopedia of Mathematics. *Associativity* [online]. Springer Verlag GmbH, European Mathematical Society. [cit. 25.4.2019]. Dostupné z `https://www.encyclopediaofmath.org/index.php/Associativity`

[10] Encyclopedia of Mathematics. *Distributivity* [online]. Springer Verlag GmbH, European Mathematical Society. [cit. 25.4.2019]. Dostupné z `https://www.encyclopediaofmath.org/index.php/Distributivity`

[11] University of Massachusetts Amherst. *Lecture 13: Quantifier Laws* [online]. University of Massachusetts Amherst, 2005. [cit. 25.4.2019]. Dostupné z `http://people.umass.edu/partee/409/Lecture%2013%20Laws%20of%20Quants.pdf`

[12] Encyclopedia of Mathematics. *De Morgan laws* [online]. Springer Verlag GmbH, European Mathematical Society. [cit. 25.4.2019]. Dostupné z `https://www.encyclopediaofmath.org/index.php/De_Morgan_laws`

[13] Northwestern University. *1.2. Quantifiers* [online]. Northwestern University. it. 25.4.2019]. Dostupné z https://sites.math.northwestern.edu/~mlerma/courses/cs310-04w/notes/dm-quantifiers.pdf

[14] Encyclopedia of Mathematics. *Prenex formula* [online]. Springer Verlag GmbH, European Mathematical Society. [cit. 25.4.2019]. Dostupné z https://www.encyclopediaofmath.org/index.php/Prenex_formula

[15] PFAHRINGER B. *Conjunctive Normal Form* [online]. Encyclopedia of Machine Learning. Springer, Boston, MA. [cit. 25.4.2019]. Dostupné z https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_158

[16] PFAHRINGER B. *Disjunctive Normal Form* [online]. Encyclopedia of Machine Learning. Springer, Boston, MA. [cit. 25.4.2019]. Dostupné z https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_223

[17] Encyclopædia Britannica. *The Predicate Calculus* [online]. Encyclopædia Britannica, inc.. Poslední změna 2.11.2018. [cit. 25.4.2019]. Dostupné z https://www.britannica.com/topic/formal-logic/The-predicate-calculus#ref534719

[18] Microsoft. .NET. *.NET* [online]. © Microsoft 2019. [cit. 25.4.2019]. Dostupné z https://dotnet.microsoft.com/

[19] Microsoft. .NET. *What is ASP.NET Core?* [online]. © Microsoft 2019. [cit. 25.4.2019]. Dostupné z https://dotnet.microsoft.com/learn/web/what-is-aspnet-core

[20] Microsoft. .NET. *Razor Pages* [online]. © Microsoft 2019. [cit. 25.4.2019]. Dostupné z https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/?view=aspnetcore-2.2

[21] DALRYMPLE Mary. *Lexical Functional Grammar* [online]. Centre for Linguistics and Philology. [cit. 25.4.2019]. Dostupné z http://users.ox.ac.uk/~cpgl0015/lfg.pdf

[22] *Treeflex* [online]. GitHub. [cit. 25.4.2019]. Dostupné z https://dumptyd.github.io/treeflex/

A Zoznam príloh

- /
  - PL-prilohy
    - PL.zip. . . . . Archív obsahujúci Visual Studio projekt
    - PL-azure.txt. . . . . Súbor s odkazom na aplikáciu nasadenú na Microsoft Azure
    - PL-publish.zip. . . . . Archív obsahujúci zložku pre nasadenie aplikácie